

Distribution of a class of divide and conquer recurrences arising from the computation of the Walsh–Hadamard transform[☆]

Paweł Hitczenko^{a,*}, Jeremy R. Johnson^b, Hung-Jen Huang^b

^aDepartment of Mathematics, Drexel University, Philadelphia, PA 19104, USA

^bDepartment Computer Science, Drexel University, Philadelphia, PA 19104, USA

Received 2 December 2003; received in revised form 11 August 2005; accepted 13 September 2005

Communicated by W. Szpankowski

Abstract

This paper explores the performance of a family of algorithms for computing the Walsh–Hadamard transform, a useful computation in signal and image processing. Recent empirical work has shown that the family of algorithms exhibit a wide range of performance and that it is non-trivial to determine which algorithm is optimal on a given computer. This paper provides a theoretical basis for the performance distribution. Performance is modeled by a family of recurrence relations that determine the number of instructions required to execute a given algorithm, and the recurrence relations can be used to explore the performance of the space of algorithms. The recurrence relations are related to standard divide and conquer recurrences, however, there are a variable number of recursive parts which can grow to infinity as the input size increases. Thus standard approaches to solving such recurrences cannot be used and new techniques must be developed. In this paper, the minimum, maximum, expected values, and variances are calculated and the limiting distribution is obtained.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Automated performance tuning; Performance models; Algorithm search space; Walsh–Hadamard transform; Divide and conquer recurrences; Random compositions; Martingales; Central limit theorem

1. Introduction

This work is motivated by the relatively new field of “Automated Performance Tuning” [1], where various techniques are used to automatically implement and optimize an algorithm on a specific computer platform. For many important algorithms, there is a large number of variations and implementation choices, and these choices can greatly affect performance. Moreover, the optimal choice among the variations and implementations is highly dependent on the underlying

[☆] The work of Paweł Hitczenko was supported in part by NSA Grant MSPF-02G-043. Part of the research of this author was carried out while he was visiting the University of Jyväskylä, Finland in the Fall of 2002. He would like to thank the Department of Mathematics and Statistics for the invitation, and Christel and Stefan Geiss for their hospitality. The work of Jeremy Johnson was supported by DARPA through research grants DABT63-98-1-0004 and NBCH1050009 administered by the Army Directorate of Contracting and NSF grant #032568.

* Corresponding author. Tel.: +1 2158952622; fax: +1 2158951582.

E-mail address: phitczenko@cs.drexel.edu (P. Hitczenko).

computer architecture and the compiler and compiler flags used to create the executable program. A particularly useful strategy for automated performance tuning, called generate-and-test, creates alternative implementation choices and uses intelligent search to find the best implementation on a given platform. This approach has been used effectively in the implementation and optimization of linear algebra [4,6,28] and signal processing [11,17,22,23] kernels.

Since this approach utilizes empirical run times, it is not necessary to have a model of the underlying computer architecture nor is it necessary to understand why a particular choice leads to good performance and another choice leads to poor performance. However, search based on empirical run times, can take a significant amount of time, and the lack of an analytic model leaves the significance of the optimization problem unclear. With better understanding, the search might prove unnecessary. The goal of this paper is to provide the first step towards a performance model and an analytic understanding of the search space for one particular problem where there is a large space of alternative algorithm variations and generate-and-test has proved successful.

The problem addressed in this paper is the computation of the Walsh–Hadamard transform (WHT). The WHT is a transform used in signal and image processing and coding theory [2,7,20]. The WHT is similar to the discrete Fourier transform (DFT) and can be computed with a divide and conquer algorithm similar to the well-known fast Fourier transform (FFT). There is a large family of fast, i.e., $\Theta(N \lg(N))$, algorithms for computing the WHT, which despite having the same arithmetic complexity, can exhibit widely varying performance. The package described in [17] can be used to explore the performance of these algorithms and can automatically select an algorithm, using generate-and-test, with good performance on a given platform. Empirically it was shown that there is a wide range of performance, and the best algorithms are rare. Various comments were given, having to do with code structure and the underlying architecture, to suggest explanations for the distribution of run times and the selection of the best algorithms on different architectures.

In this paper, a performance model is used to explore the space of WHT algorithms. The performance model, which uses instruction count, does not account for important features of modern computer architectures such as cache, pipelining, and instruction level parallelism, and thus cannot be used to predict actual performance. Nonetheless, the model does provide important insight into the performance of WHT algorithms and provides an explanation as to why there should be a wide distribution in run times. More importantly the model can be analyzed analytically and provides a theoretical understanding for the observed distributions of run times. Using the instruction count model, an analytic solution is provided to the optimization problem, the average instruction count is calculated, and the limiting distribution is determined. These results are obtained through the study of a class of divide and conquer recurrences.

The divide and conquer algorithms for computing the WHT arise from arbitrary compositions (i.e., ordered partitions) of the exponent n of the size $N = 2^n$ of the transform. A particular divide and conquer strategy is determined by a composition of n , and the recursive computations corresponding to the parts of the composition are recursively determined in the same fashion. A particular algorithm is specified by a tree corresponding to the recursive selection of compositions, and the number of instructions required to execute the algorithm can be determined from the tree through a set of recurrence relations. The optimization problem investigated is to find the tree of size n corresponding to the WHT algorithm with the fewest instructions. The performance distribution for the space of algorithms is modeled by the distribution of instruction counts for randomly selected algorithms, where a random algorithm is chosen by recursively selecting random compositions for each node in the tree.

Typically, divide and conquer is applied by decomposing a problem of a given size n into a small (typically 2), deterministic number of subproblems of smaller sizes. Our partitioning strategies, allowing search over *all* compositions of n are a sharp departure from a standard scheme: not only is the number of subproblems random, it is also allowed to grow to infinity with n (and typically does). This leads, in principle, to more complex recurrences. While there is a vast literature (see, e.g., [21,15, and references therein]) on traditional divide and conquer recurrences (often referred to as quicksort type recurrences), the more general scheme has been addressed only sporadically. Certain cases are considered in [9, Section 2] under the name stochastic divide and conquer recurrences while Roura [26, Section 3] uses the name continuous divide and conquer recurrences. These are generally recurrences of the form

$$f_n = \sum_{k=1}^{n-1} \xi_{n,k} f_k + t_n,$$

where the coefficients $(\xi_{n,k})$ and a sequence (t_n) are known. Flajolet et al. considered the case $\xi_{n,k} = (n-k)/(n(n+1))$. Roura treated a more general case of (essentially polynomial) coefficients under some regularity assumptions. Our

consideration of expected values in Section 5.3 (see, Eq. (16)) lead to the same type of recurrence with $\xi_{n,k} = (n+3-k)2^{-k-1}$, up to a constant only dependent on n , which we solve by elementary means. In view of these facts, however, it seems that would be worthwhile to try to develop a fairly general theory of such recurrences, with as mild assumptions on $\xi_{n,k}$'s as possible.

As for the issue of limiting distribution, Régnier [24] proved by martingale methods that the normalized number of comparisons in quicksort algorithm converges in distribution and Rösler [25] used Banach's fixed point theorem to characterize the limiting distribution of quicksort as the *unique* solution of an underlying distributional equation. Rösler's approach, now referred to as contraction method proved very useful in analyzing other divide and conquer recurrences (see, e.g., examples and references in [15]) as well as in studying more subtle properties of the limiting distribution of quicksort (see, for example, [8]). Of crucial importance, however, is the fact the each recursive subdivision is into two (or a number bounded independently of the size n) subproblems, an assumption that fails in our case. Consequently, the methods developed for studying quicksort type recurrences are of little use in this context. Our approach is via martingale techniques (albeit quite different than Régnier's); the main tool is the central limit theorem for martingales.

The paper is organized as follows. Section 2 reviews the Walsh–Hadamard transform and defines the space of algorithms for computing it that will be considered. Section 3 presents the instruction count model, and Section 4 summarizes empirical data that illustrates the instruction count model and properties of the space of WHT algorithms. Section 5 analyzes a family of divide and conquer recurrences that arise when the performance model is applied to the family of WHT algorithms. The minimum, maximum, expected values, and variances are calculated and the limiting distribution is obtained. These results, when restricted to binary splits, contain various known results as special cases. Section 6 compares the special cases of the general result to these known results.

2. The Walsh–Hadamard transform

The Walsh–Hadamard transform of a signal x , of size $N = 2^n$, is the matrix-vector product $\mathbf{WHT}_N \cdot x$, where

$$\mathbf{WHT}_N = \bigotimes_{i=1}^n \mathbf{DFT}_2 = \overbrace{\mathbf{DFT}_2 \otimes \cdots \otimes \mathbf{DFT}_2}^n.$$

The matrix

$$\mathbf{DFT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

is the 2-point DFT matrix, and \otimes denotes the tensor or Kronecker product. The tensor product of two matrices is obtained by replacing each entry of the first matrix by that element multiplied by the second matrix. Thus, for example,

$$\begin{aligned} \mathbf{WHT}_4 &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \end{aligned}$$

Algorithms for computing the WHT can be derived using properties of the tensor product [27,18]. More precisely, algorithms for computing the WHT can be represented by structured factorizations of the WHT matrix, and such factorizations can be derived using the multiplicative property of tensor products: $(A \otimes B)(C \otimes D) = AC \otimes BD$. For example, a recursive algorithm for the WHT is obtained from the factorization

$$\mathbf{WHT}_{2^n} = (\mathbf{WHT}_2 \otimes I_{2^{n-1}})(I_2 \otimes \mathbf{WHT}_{2^{n-1}}), \quad (1)$$

where I_m is the $m \times m$ identity matrix. An algorithm to compute $y = \mathbf{WHT}_{2^n}x$, corresponding to this factorization, is obtained by first computing $t = (I_2 \otimes \mathbf{WHT}_{2^{n-1}})$, which involves two recursive calls to compute $\mathbf{WHT}_{2^{n-1}}$, and

then computing $y = (\mathbf{WHT}_2 \otimes I_{2^{n-1}})t$ which is computed by applying \mathbf{WHT}_2 to subvectors of t containing the pair of elements $(t_i, t_{i+2^{n-1}})$, for $i = 0, \dots, 2^{n-1} - 1$. Note that this factorization does not specify how the recursive calls are computed. This would be determined by a recursive factorization of $\mathbf{WHT}_{2^{n-1}}$.

An iterative algorithm for computing the WHT is obtained from the factorization

$$\mathbf{WHT}_{2^n} = \prod_{i=1}^n (I_{2^{i-1}} \otimes \mathbf{WHT}_2 \otimes I_{2^{n-i}}), \tag{2}$$

which can be proven (see, [18]) using induction and the multiplicative property along with the identity $I_m \otimes I_n = I_{mn}$. More generally, let $n = n_1 + \dots + n_t$, then

$$\mathbf{WHT}_{2^n} = \prod_{i=1}^t (I_{2^{n_1+\dots+n_{i-1}}} \otimes \mathbf{WHT}_{2^{n_i}} \otimes I_{2^{n_{i+1}+\dots+n_t}}). \tag{3}$$

This equation encompasses both the iterative and recursive algorithm and provides a mechanism for exploring different breakdown strategies and combinations of recursion and iteration.

Let $N = N_1 \cdot \dots \cdot N_t$, where $N_i = 2^{n_i}$, and let $x_{b,s}^M$ denote the vector $(x(b), x(b+s), \dots, x(b+(M-1)s))$. Then evaluation of $\mathbf{WHT}_N \cdot x$ using Eq. (3) is performed using

```

R = N;  S = 1;
for i = 1, ..., t,
    R = R/N_i;
    for j = 0, ..., R - 1,
        for k = 0, ..., S - 1,
             $x_{jN_i S+k, S}^{N_i} = \mathbf{WHT}_{N_i} \cdot x_{jN_i S+k, S}^{N_i}$ ;
    S = S * N_i.
    
```

The computation of \mathbf{WHT}_{N_i} is computed recursively in a similar fashion until a base case of the recursion is encountered. Observe that \mathbf{WHT}_{N_i} is called N/N_i times. Small WHT transforms are computed using the same approach; however, the code is unrolled in order to avoid the overhead of loops or recursion. This scheme assumes that the algorithm works in-place and is able to accept stride parameters.

Alternative algorithms are obtained through different sequences of the application of Eq. (3). Each algorithm obtained this way can be represented by a tree, called a partition tree. The root of the partition tree corresponding to an algorithm for computing \mathbf{WHT}_N , where $N = 2^n$ is labeled with n . Each application of Eq. (3) corresponds to an expansion of a node into children whose sum equals the node. Leaf nodes in the tree correspond to the base case of the recursion. A node labeled with m corresponds to the computation of \mathbf{WHT}_{2^m} . If the node is in a tree rooted with n , the computation of \mathbf{WHT}_{2^m} is performed 2^{n-m} times. Fig. 1 shows the trees for an iterative and recursive algorithm for computing \mathbf{WHT}_{16} .

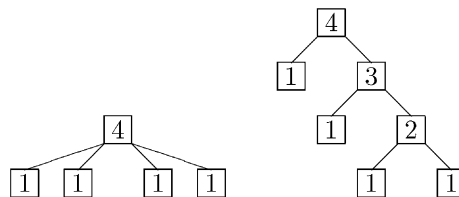


Fig. 1. Partition trees for iterative and recursive WHT algorithms.

Table 1
Number of partition trees for **WHT**_{2n}

<i>n</i>	1	2	3	4	5	6	7	8
<i>T</i> _{<i>n</i>}	1	2	6	24	112	568	3032	16 768
\tilde{T} _{<i>n</i>}	1	1	3	11	45	197	903	4279
<i>B</i> _{<i>n</i>}	1	2	5	15	51	188	731	2950
\tilde{B} _{<i>n</i>}	1	1	2	5	14	42	132	429

In this paper, the performance of WHT algorithms corresponding to all possible partition trees and various subsets of partition trees is explored. The total number of partition trees of size *n* is given by the recurrence

$$T_n = 1 + \sum_{n_1+\dots+n_k=n} T_{n_1} \cdots T_{n_k}, \tag{4}$$

and the subset of fully expanded partition trees (i.e., all leaves equal to 1) satisfies the recurrence

$$\tilde{T}_n = \sum_{n_1+\dots+n_k=n} \tilde{T}_{n_1} \cdots \tilde{T}_{n_k}. \tag{5}$$

The number of binary and fully expanded binary partition trees satisfy the recurrences

$$B_n = 1 + \sum_{n_1+n_2=n} B_{n_1} \cdot B_{n_2}, \tag{6}$$

and

$$\tilde{B}_n = \sum_{n_1+n_2=n} \tilde{B}_{n_1} \cdot \tilde{B}_{n_2}. \tag{7}$$

Table 1 lists the first few values of *T*_{*n*}, \tilde{T} _{*n*}, *B*_{*n*}, and \tilde{B} _{*n*}.

Note that (7) is the recurrence for Catalan numbers and thus

$$\tilde{B}_n = \frac{1}{n} \binom{2(n-1)}{n-1} \sim \frac{4^n}{4\sqrt{\pi n^{3/2}}}.$$

The generating function, $\tilde{B}(z)$, for \tilde{B}_n satisfies the functional equation

$$\tilde{B}(z) = z + \tilde{B}(z)^2.$$

The other recurrences satisfy similar functional equations: $B(z) = z/(1-z) + B(z)^2$, $\tilde{T}(z) = z + \tilde{T}(z)^2/(1-\tilde{T}(z))$, and $T(z) = z/(1-z) + T(z)^2/(1-T(z))$. These equations are algebraic of degree 2 and are amenable to the methods described in [10]. Specifically, all four generating functions have the square root singularity and can be treated by the positive implicit functions theorem (see, [10, Theorem VII.3]). It follows that the number of binary trees is $B_n \sim c_B 5^n/n^{3/2}$, the number of fully expanded trees is $\tilde{T}_n \sim c_{\tilde{T}} \beta^n/n^{3/2}$, where $\beta = 3 + 2\sqrt{2} \approx 5.828427120$, and the number of all partition trees is $T_n \sim c_T \alpha^n/n^{3/2}$, where $\alpha = 4 + 2\sqrt{2} \approx 6.828427120$. The values of the constants c_B , $c_{\tilde{T}}$, and c_T are

$$c_B = \frac{\sqrt{5}}{8\sqrt{\pi}}, \quad c_{\tilde{T}} = \frac{1}{2\sqrt{\pi}} \sqrt{\frac{3-2\sqrt{2}}{2\sqrt{2}}}, \quad c_T = \frac{\sqrt{\sqrt{2}-1}}{2(1+\sqrt{2})\sqrt{\pi}}.$$

3. Performance model for the WHT

In the previous section it was shown that there is a large family of WHT algorithms which have varying degrees of recursion, iteration, and straight-line code. A natural question is to determine which algorithm leads to the best

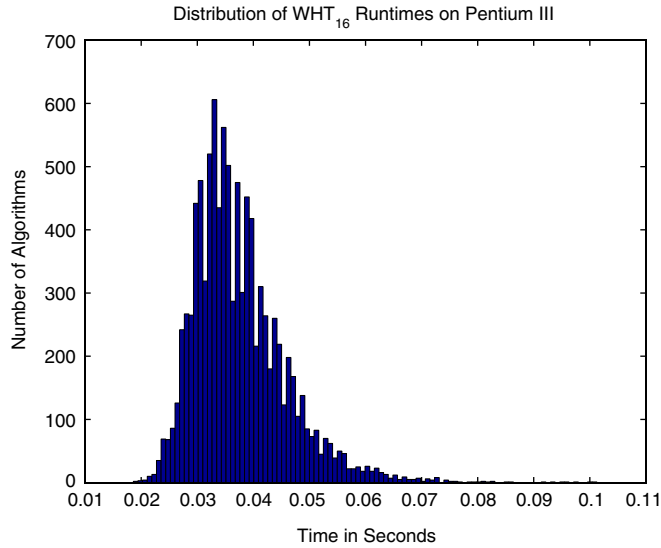


Fig. 2. Performance histogram on the Pentium III.

performance. The histogram in Fig. 2 shows that there is in fact a wide range in performance. The times in Fig. 2 were obtained using the WHT package from [17], and were obtained on a Pentium III with 128 MB of memory running at 550 MHz. The histogram shows the runtimes for 10,000 randomly generated WHT algorithms of size 2^{16} .

The wide range of times in Fig. 2 are not due to the number of arithmetic operations. The following theorem shows that all algorithms have exactly the same number of floating point operations (flops).

Theorem 1. *Let \mathcal{W}_N be a fully expanded WHT algorithm for computing \mathbf{WHT}_N . Then $\text{flops}(\mathcal{W}_N) = N \lg(N)$.*

Here and throughout the paper \lg denotes \log_2 . The proof is by induction on $N = 2^n$. The base case for $n = 1$ is clearly true. In general, assume that \mathcal{W}_N uses the factorization

$$\prod_{i=1}^t (\mathbf{I}_{2^{n_1+\dots+n_{i-1}}} \otimes \mathcal{W}_{2^{n_i}} \otimes \mathbf{I}_{2^{n_{i+1}+\dots+n_t}}),$$

where $\mathcal{W}_{2^{n_i}}$ is an algorithm to compute $\mathbf{WHT}_{2^{n_i}}$. Since $\mathcal{W}_{2^{n_i}}$ is called 2^{n-n_i} times,

$$\text{flops}(\mathcal{W}_N) = \sum_{i=1}^t 2^{n-n_i} \text{flops}(\mathcal{W}_{2^{n_i}}),$$

which by induction is equal to

$$\sum_{i=1}^t 2^{n-n_i} 2^{n_i} n_i = N \sum_{i=1}^t n_i = Nn = N \lg(N).$$

Since arithmetic operations cannot distinguish algorithms, the distribution of runtimes must be due to other factors. In [14], other performance metrics, such as instruction count, memory accesses, and cache misses, were gathered and their influence on runtime was investigated. Different algorithms can have vastly different instruction counts due to the varying amounts of control overhead from recursion, iteration, and straight-line code. Different algorithms access data in different patterns, with varying amounts of locality. Consequently, different algorithms can have different amounts of instruction and data cache misses.

In this paper, the focus is on an instruction count model and the mathematical techniques required to analyze the number of instructions for different WHT algorithms. While instruction count does not accurately model performance, it is an important aspect of performance and the model provides insight into many aspects of the performance tradeoff

Table 2
Instruction constants for the Pentium III using gcc version 2.91.66

α	α_1	α_2	α_3	β_1	β_2	β_3
27	12	34	106	18	18	20

of different WHT algorithms. Furthermore, the model is sufficiently complex to exhibit non-trivial behavior yet is amenable to analytic techniques.

The number of instructions required by an algorithm depends on the way the algorithm is implemented, the compiler that translates the program implementing the algorithm into machine instructions, and the machine on which the algorithm is executed. It is possible to derive a set of parameterized recurrence relations for the number of machine instructions used by an arbitrary WHT algorithm. The parameters depend on the program, compiler and machine used.

The WHT package executes code similar to the pseudo-code in the previous section each time the algorithm is called recursively. When a leaf node is encountered a procedure to compute the corresponding WHT, implemented using straight-line code, is called. A library of small WHT procedures is generated by a family of code generators. In the WHT package straight-line code is considered only for sizes 2^n for $n = 1, \dots, 8$, since it has been determined (see, [17,14]) that straight-line code for larger sizes is not beneficial on current computers.

In order to count the number of instructions, it is necessary to know how many times the recursive WHT algorithm is called, how many times each loop body is executed, and how many times each small WHT procedure is called. Given this information and the number of instructions for the straight-line code and the basic blocks in the WHT procedure, it is possible to determine a formula for the number of instructions. For a particular WHT algorithm \mathcal{W}_{2^n} let $A(n)$ be the number of times the recursive WHT procedure is called, $A_l(n)$ the number of times the straight-line code for \mathbf{WHT}_{2^l} is called, $L_1(n)$ the number of times the outer loop is executed, $L_2(n)$ the number of times the middle loop is executed, $L_3(n)$ the number of times the innermost loop is executed. Then the number of instructions required to execute \mathcal{W}_{2^n} is equal to

$$\alpha A(n) + \sum_{l=1}^8 \alpha_l A_l(n) + \sum_{i=1}^3 \beta_i L_i(n),$$

where α is the number of instructions for the code in the compiled WHT procedure executed outside the loops, α_i , $i = 1, \dots, 8$ is the number of instructions in the compiled straight-line code implementations of small WHT's of size 1–8, and β_i , $i = 1, 2, 3$ is the number of instructions executed in the outer-most, middle, and inner-most loops in the compiled WHT procedure.

The α and β constants are determined by examining the assembly code produced by compiling the WHT package. Table 2 shows the values obtained for the Pentium III using the gcc compiler version 2.91.66 with flags set to “-O6 -fomit-frame-pointer-pedantic-malign-double-Wall”.

The functions $A(n)$, $A_l(n)$, $L_i(n)$ satisfy recurrences that can be derived from the pseudo code in Section 3.

The WHT procedure is called once plus the number of calls in $\mathcal{W}_{2^{n_i}}$ for $i = 1, \dots, t$, and since $\mathcal{W}_{2^{n_i}}$ is called 2^{n-n_i} , the number of recursive calls to the WHT procedure in \mathcal{W}_{2^n} is equal to

$$A(n) = \begin{cases} 1 + \sum_{i=1}^t 2^{n-n_i} A(n_i) & \text{if } n = n_1 + \dots + n_t, \\ 0 & \text{if } n \text{ is a leaf.} \end{cases} \quad (8)$$

Similarly the number of calls to the straight-line code for \mathbf{WHT}_{2^l} in the algorithm \mathcal{W}_{2^n} is equal to

$$A_l(n) = \begin{cases} \sum_{i=1}^t 2^{n-n_i} A_l(n_i) & \text{if } n = n_1 + \dots + n_t, \\ 1 & \text{if } n = l \text{ is a leaf.} \end{cases} \quad (9)$$

It is easy to show that $A_l(n) = v_l 2^{n-l}$, where v_l is the number of leaf nodes with value l in the tree corresponding to \mathcal{W}_n .

Since the outermost loop is executed t times, the middle loop is executed $2^{n_1+\dots+n_{i-1}}$ times for the i th execution of the outermost loop, and the innermost loop is executed 2^{n-n_i} times for each iteration of the middle loop in the i th

iteration of the outer loop,

$$L_1(n) = \begin{cases} t + \sum_{i=1}^t 2^{n-n_i} L_1(n_i) & \text{if } n = n_1 + \dots + n_t, \\ 0 & \text{if } n \text{ is a leaf,} \end{cases} \tag{10}$$

$$L_2(n) = \begin{cases} \sum_{i=1}^t \{2^{n-n_i} L_2(n_i) + 2^{n_1+\dots+n_{i-1}}\} & \text{if } n = n_1 + \dots + n_t, \\ 0 & \text{if } n \text{ is a leaf,} \end{cases} \tag{11}$$

$$L_3(n) = \begin{cases} \sum_{i=1}^t \{2^{n-n_i} L_3(n_i) + 2^{n-n_i}\} & \text{if } n = n_1 + \dots + n_t, \\ 0 & \text{if } n \text{ is a leaf.} \end{cases} \tag{12}$$

4. Empirical observations of the performance model

This section presents empirical data using the instruction count model with constants set to the values in Table 2. Observations from the data provide some insight into the behavior of the family of WHT algorithms presented in Section 2. Furthermore, the data suggests theorems which will be proved in the following sections.

Table 3 compares the number of instructions used by four families of WHT algorithms. Counts are reported as a ratio of the number of instructions used by a given algorithm of specified size to the number of instructions used by the iterative algorithm of the same size. The iterative algorithm is obtained by setting $n = 1 + \dots + 1$ in Eq. (3). The *left recursive*, *right recursive*, and *balanced* algorithms are obtained by recursively splitting n using $n = 1 + (n - 1)$, $n = (n - 1) + 1$, and $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$ in Eq. (3), respectively.

In all cases the iterative algorithm has the fewest number of instructions. Note that the number of instructions used by the right and left recursive algorithms differs. This difference is due solely to the L_2 component corresponding to the middle loop, as all other cost functions are invariant under permutations of the children. Moreover, the L_2 recurrence is lowest if the larger children are to the right.

The data in Table 3 suggest that there are limiting ratios. This can be verified by specializing recurrences 8–12 to the algorithms in the table. For all of the algorithms $A_1(n) = n2^{n-1}$. For the iterative algorithm $A(n) = 1$, $L_1(n) = n$, $L_2(n) = 2^n - 1$, and $L_3(n) = n2^{n-1}$. For the right recursive algorithm, $A(n) = 2^n - 1$, $L_1(n) = 2(2^n - 1)$, $L_2(n) = 3(2^n - 1)$, and $L_3(n) = n2^{n-1} + 2^{n+1} - 2$. The left recursive algorithm has the same solutions to the recurrence relations except $L_2(n) = n2^{n-1} + 2^n - 1$. A simple closed form solution to the recurrences for the balanced algorithm was not found; however, $A(n)$ and $L_i(n)$ for $i = 1, 2, 3$ are all $\Theta(n2^n)$ and the limiting constants can be determined numerically. Specifically, we have $A(n) = \rho_0 n2^n$ and, for $i = 1, 2, 3$, $L_i(n) = \rho_i n2^n$ where $\rho_0 = 0.1411142349\dots$, $\rho_1 = 0.2822284699\dots$, $\rho_2 = 0.4616713524\dots$, and $\rho_3 = 0.6411142349\dots$. Plugging in the constants for the Pentium III the limiting ratios in Table 3 are 1, 1.5625, and 2.251410365, respectively.

Table 3
Ratio of instruction counts recursive, balanced, and iterative algorithms

Size	Right recursive/iterative	Left recursive/iterative	Balanced/iterative
2	1.00	1.00	1.00
3	1.31	1.37	1.37
4	1.41	1.54	1.63
5	1.42	1.61	1.75
6	1.40	1.64	1.82
7	1.37	1.64	1.91
8	1.34	1.64	1.97
9	1.31	1.64	1.99
10	1.28	1.64	2.00
11	1.26	1.63	2.01
12	1.24	1.63	2.02
13	1.22	1.62	2.04
14	1.21	1.62	2.07
15	1.20	1.61	2.09
16	1.18	1.61	2.10

The data in Table 3 and the ensuing discussion suggests that the iterative algorithm is optimal and that some combination of a balanced tree favoring larger children to the right is the worst case. Moreover it appears, for fully expanded trees using the Pentium III constants, that there is a factor of about two between the best and worst performance in terms of instruction counts.

The trees with the minimum and maximum instruction counts can be found using dynamic programming, since using the instruction count model, the optimal tree of a given size is independent of the context in which it is called (i.e., where in the tree it is located). Dynamic programming is applied by generating all possible splits of a node of size n and computing the max or min value using the max/min values for each of the children. The values for all smaller sizes must be computed and once they are computed the values are obtained by table lookup. To implement this procedure it is necessary to generate all possible splits. This is done using a one-to-one mapping between compositions of n and $(n - 1)$ -bit numbers. The mapping is obtained by considering a string of n ones and placing the bits of the $(n - 1)$ -bit binary number between adjacent ones. The string of ones are summed until a 1-bit is encountered (assume an implicit 1-bit at the end). For example, the composition generated from the number $j = 0\ 100\ 110$ is $[2, 3, 1, 2]$.

This procedure was implemented using Maple version 8 and the Pentium III constants. The optimal and worst case trees were obtained for size $n = 2, \dots, 16$. For all sizes, the iterative algorithm using leaf nodes of size 3 was optimal (2 was also used when n is not a multiple of 3). More precisely, when $n \equiv 0 \pmod{3}$, the iterative tree with $n/3$ children of size 3 is optimal, when $n \equiv 1 \pmod{3}$, the iterative tree with $\lfloor n/3 \rfloor - 1$ children of size 3 and two leftmost children of size 2 is optimal, and when $n \equiv 2 \pmod{3}$, the iterative tree $\lfloor n/3 \rfloor$ children of size 3 and the left child of size 2 is optimal. The function $A_I(n)$ which counts the number of instructions due to leaf nodes can be used to suggest which leaves will occur in the optimal tree. First note that all trees with the same set of leaf nodes have the same value of $A_I(n)$. When comparing two iterative trees the contribution due to all leaf nodes with the same value can be ignored. Thus, for example, in the case when $n \equiv 1 \pmod{3}$, it is required only to compare $2^{n-2}\alpha_2 + 2^{n-2}\alpha_2$ with $2^{n-1}\alpha_1 + 2^{n-3}\alpha_3$. In this case, the tree with two leaves of size 2 will be chosen over the tree with one leaf of size 1 and one leaf of size 3 when $4\alpha_2 \leq 4\alpha_1 + \alpha_3$, as is the case for the Pentium III.

Table 4 compares the best WHT algorithm with the worst. It shows that a factor of 6–7 is available by choosing the appropriate algorithm. Figs. 3 and 4 show the trees that lead to the maximum instruction counts for size 13 and 16. The tree of size 13 is an example of a balanced power of two tree [16]. A balanced power of two trees of size n is the binary tree whose children are of size 2^k and $n - 2^k$, where 2^k is chosen to be the nearest power of two to $\lfloor n/2 \rfloor$ (when $\lfloor n/2 \rfloor$ is equidistant from two powers of two the choice is arbitrary since $n - 2^k$ will be the other choice). This procedure is applied recursively and the larger of the two children is selected as the left child. Note that the worst case tree in Fig. 4 is not a balanced power of two trees. It will be shown in Section 5.2 that the recurrences for $A(n)$, $L_1(n)$, and $L_3(n)$, when the input trees are fully expanded, are maximized when the input tree is a balanced power of two tree; however, $L_2(n)$ is maximized when the input tree is left recursive. Thus the tree with the maximum instruction count will depend on the particular machine constants used, with some weight being given to leftmost trees and some weight given to balanced trees.

If the space of algorithms is restricted to binary trees the optimal algorithm will be somewhat worse than the optimal algorithm in the entire space of WHT algorithms. However, it was shown when comparing the fully expanded recursive algorithm to the fully expanded iterative algorithm, that in the limit the ratio of instruction counts approaches one. Table 5 shows the ratio of the optimal binary algorithm to the optimal algorithm. The optimal binary algorithm is a right recursive algorithm with the same sequence of leaf nodes as the optimal iterative algorithm.

By comparing the maximum and the minimum number of instructions it is possible to determine the range in performance; however, it would be useful to know how far from the optimal typical trees are and more generally what

Table 4
Ratio of the worst to the best WHT algorithm using Pentium III instruction counts

n	2	3	4	5	6	7	8	9
Ratio max/min	7.21	7.65	3.89	5.21	6.15	5.40	6.01	6.45
n	10	11	12	13	14	15	16	
Ratio max/min	5.88	6.23	6.60	6.08	6.38	6.63	6.23	

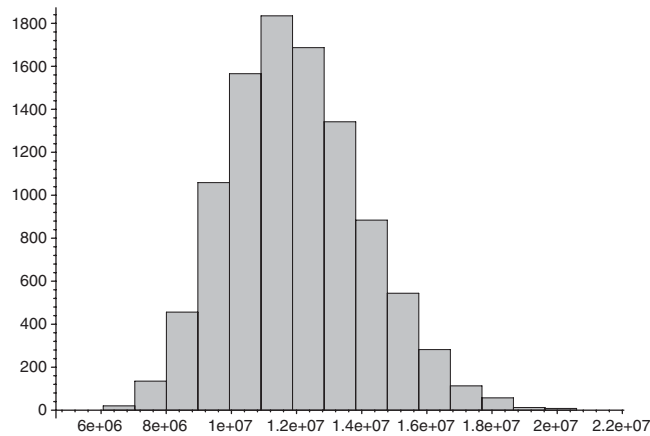


Fig. 5. Instruction count histogram on the Pentium III.

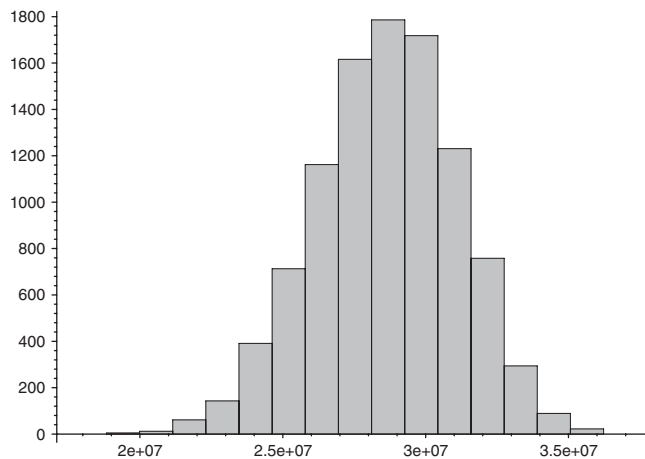


Fig. 6. Instruction count histogram for fully expanded trees on the Pentium III.

instructions, determined by dynamic programming, is 6 066 945 and the maximum number of instructions is 37 817 249. The minimum number of instructions obtained in the sample is 6 088 617, while the maximum is 20 616 167.

If the randomly generated trees are restricted to fully expanded trees, then the distribution more closely resembles a normal distribution (see, Fig. 6). The mean is approximately 2.862×10^7 and the variance is approximately 6.147×10^{12} . The minimum number of instructions obtained in the sample is 18 827 489, while the maximum is 36 226 849.

5. Analysis of WHT recurrences

In this section we establish theoretical results concerning recurrences 8 and 10–12 derived in Section 3 and we will see that they agree with empirical observations discussed in Section 4.

5.1. Mathematical model

For the purpose of mathematical analysis we concentrate on fully expanded partition trees (i.e., all leaves equal to 1). This assumption does not change the nature of general phenomenon; it simplifies analysis a little bit and, most importantly, makes it cleaner. Of course, as illustrated in Section 4 from practical point of view it is important to consider leaves of larger sizes as well.

Consider first recurrence 8. After dividing by 2^n and letting $L_0(k) = A(k)/2^k$, for $k \geq 2$ it becomes

$$L_0(n) = \frac{1}{2^n} + \sum_{i=1}^t L_0(n_i).$$

For future considerations, it is convenient to look at the slightly more general setting. Namely, the above relation may be viewed as

$$L_0(n) = \sum_{i=1}^t L_0(n_i) + T_0(n),$$

where the sequence $T_0(n)$ (which in this case is just $1/2^n$) may be more general. It is usually referred to as a toll function and we will allow it to depend not only on n but also on the composition (n_1, \dots, n_t) .

As for the other recurrences, for $L_3(n)$ add 1 to both sides and then divide through by 2^n to see that the quantity $(L_3(n) + 1)/2^n$ satisfies exactly the same recurrence as $L_0(n)$ and thus will not be of interest anymore. For the remaining two, divide through by 2^n . The corresponding toll $T_1(n)$ is

$$T_1(n) = \frac{t}{2^n}.$$

Finally, for $L_2(n)$ we obtain

$$\frac{L_2(n)}{2^n} = \sum_{i=1}^t \left\{ \frac{L_2(n_i)}{2^{n_i}} + \frac{1}{2^{n_i+n_{i+1}+\dots+n_t}} \right\},$$

i.e., the toll function in this case is

$$T_2(n) = \sum_{i=1}^t \frac{1}{2^{n_i+n_{i+1}+\dots+n_t}}.$$

This last toll function explains why it is appropriate to use compositions rather than partitions (i.e., unordered compositions) in our model; T_2 depends not only on the sizes of n_1, \dots, n_t but also on their order. Letting $F_k(n) = L_k(n)/2^n$ for $k = 0, 1, 2$, we see that all F 's satisfy the same recurrence with different toll functions. Thus, we will consider a generic recurrence of the form

$$F(n) = \sum_{i=1}^t F(n_i) + T(n), \tag{13}$$

only occasionally referring to the specific toll function $T_k(n)$ or the recurrence $F_k(n)$.

We wish to analyze the limiting distribution of the random variable $F(n)$ under the assumption that a composition (n_1, \dots, n_t) is chosen uniformly at random from the set Ω_n of $2^n - 1$ compositions of n into at least two parts. We recall [12,13] that if all $2^n - 1$ compositions are considered, then a randomly chosen composition is equidistributed with

$$\left(\Gamma_1, \Gamma_2, \dots, \Gamma_{\tau-1}, n - \sum_{j=1}^{\tau-1} \Gamma_j \right), \tag{14}$$

where $\Gamma_1, \Gamma_2 \dots$ are i.i.d. geometric random variables with parameter $\frac{1}{2}$, $\text{GEOM}(\frac{1}{2})$. That is

$$\Pr(\Gamma_1 = j) = \frac{1}{2^j}, \quad j = 1, 2, \dots,$$

and τ is a stopping time defined by

$$\tau = \inf\{k \geq 1 : \Gamma_1 + \Gamma_2 + \dots + \Gamma_k \geq n\}. \tag{15}$$

Before we proceed any further we would like to stress that this model is not the same as an alternative in which a partition tree of size n is chosen uniformly at random among *all* partition trees of that size. For example, if fully expanded partition trees are considered, then under the ‘‘uniform choice among all trees’’ model, each of such trees

is selected with probability $1/\tilde{T}_n$ (see, (5)). Under our model, however, the probabilities vary from tree to tree. For example, an iterative tree is selected with probability $1/(2^{n-1} - 1)$ while the same probability for a recursive tree is only

$$\prod_{j=1}^{n-1} \frac{1}{2^{j-1} - 1}.$$

Our model, which assumes that at every partitioning stage a node is split uniformly over all non-trivial possibilities and this is done independently of all other splitting decisions, reflects a recursive design of WHT algorithms.

When a random model is adopted, the $F(n)$'s and $T(n)$'s become random variables, and the equality in Eq. (13) is understood to hold in distribution. According to our assumptions $F(n_i)$'s depend on a particular composition only through the sizes of parts n_i . That is, $F(n_i)$'s are conditionally independent once the composition is chosen. The tolls $T(n)$ depend on the sizes n and their compositions, but not on F .

We will assume that the initial value $F(1)$ is a given nonrandom number. Since the calls to leaf nodes are treated separately by recurrence (9), the initial condition for each of the recurrences is $F(1) = 0$, but mathematically it makes no difference if the initial value is set to be any other number.

Let us begin by establishing deterministic bounds on these recurrences.

5.2. Deterministic bounds

In order to state our bounds we need a bit more notation. For a binary partition tree T with internal nodes $I(T)$ we let

$$w(T) = \sum_{x \in I(T)} \frac{1}{2^x}.$$

Let us consider the so-called balanced power of two tree (see, [16]) that is a partition tree that can be defined as follows: given a positive integer n , consider its binary expansion

$$n = 2^{k_1} + 2^{k_2} + \dots + 2^{k_j}, \quad k_1 > k_2 > \dots > k_j.$$

The root n is split as

$$n = \begin{cases} 2^{k_1} + (n - 2^{k_1}) & \text{if } k_2 = k_1 - 1, \\ 2^{k_1-1} + (n - 2^{k_1-1}) & \text{if } k_2 < k_1 - 1. \end{cases}$$

The same rule is then applied recursively. We let $w_n = w(T_b(n))$, where $T_b(n)$ is the balanced power of two partition tree for n . We have

Proposition 2. *The following are tight bounds on the given recurrences:*

- (i) $nF_0(1) + \frac{1}{2^n} \leq F_0(n) \leq nF_0(1) + w_n,$
- (ii) $n \left(F_1(1) + \frac{1}{2^n} \right) \leq F_1(n) \leq nF_1(1) + 2w_n,$
- (iii) $nF_2(1) + 1 - \frac{1}{2^n} \leq F_2(n) \leq n(F_2(1) + 1) - \frac{1}{2^n}.$

Before proving this proposition let us remark that lower bounds are exactly the values of the respective recurrences when an iterative partition tree is used and that the upper bound for F_2 is the value attained at the left-recursive partition tree. The upper bounds for F_0 , and F_1 are the values obtained on $T_b(n)$. This value can be obtained numerically; for example, when $n = 2^m$ is a perfect power of 2 then,

$$w_n = \sum_{j=0}^{m-1} 2^j 2^{-2^{m-j}} = 2^m \sum_{j=1}^m 2^{-j} 2^{-2^j} \leq n \sum_{j=1}^{\infty} 2^{-j} 2^{-2^j} = nw,$$

where $w \sim 0.1411142349 \dots$

Proof. It is an easy inductive proof to show that the given expressions are lower bounds. For example,

$$\begin{aligned} F_0(n) &= \sum_{i=1}^t F_0(n_i) + \frac{1}{2^n} = \sum_{i: n_i \geq 2} F_0(n_i) + \sum_{i: n_i=1} F_0(n_i) + \frac{1}{2^n} \\ &\geq \sum_{i: n_i \geq 2} \left(n_i F_0(1) + \frac{1}{2^{n_i}} \right) + \sum_{i: n_i=1} F_0(1) + \frac{1}{2^n} \\ &= nF_0(1) + \frac{1}{2^n} + \sum_{i: n_i \geq 2} \frac{1}{2^{n_i}} \geq nF_0(1) + \frac{1}{2^n}. \end{aligned}$$

For (ii) we obtain

$$F_1(n) \geq nF(1) + \frac{t}{2^n} + \sum_{i: n_i \geq 2} \frac{n_i}{2^{n_i}} \geq nF(1) + \frac{\#\{i : n_i = 1\}}{2^n} + \sum_{i: n_i \geq 2} \frac{n_i}{2^{n_i}} = nF(1) + \frac{n}{2^n}.$$

For (iii) observe that once a composition $n = n_1 + \dots + n_t$ is chosen, the n_i 's should be arranged in a non-decreasing order. That is because, the sum of $F_2(n_i)$ remains the same but the sum of $1/2^{n_i+\dots+n_t}$ is minimized if the n_i 's are non-decreasing. If now this is the case, and if there is an n_i larger than 1, then, in particular, $n_t \geq 2$, and, by inductive hypothesis, $F_2(n_t) \geq n_t F_2(1) + 1 - 1/2^{n_t}$, and also $F_2(m) \geq mF_2(1)$ for $1 \leq m \leq n - 1$. Thus,

$$\begin{aligned} F_2(n) &= \sum_{i=1}^{t-1} \left(F_2(n_i) + \frac{1}{2^{n_i+\dots+n_t}} \right) + \left(\frac{1}{2^{n_t}} + F_2(n_t) \right) \\ &\geq \sum_{i=1}^{t-1} n_i F_2(1) + \frac{1}{2^{n_t}} + n_t F_2(1) + 1 - \frac{1}{2^{n_t}} \\ &= nF_2(1) + 1 \geq nF_2(1) + 1 - \frac{1}{2^n}. \end{aligned}$$

We now turn to the upper bounds. First, notice that for each recurrence, binary splits give the worst behavior. The reason is that if there are more than two parts, then merging two of them together would increase the value. For the first two any two can be merged and for the third, merging the last two parts would increase the value

$$\begin{aligned} F_2(n) &= \sum_{i=1}^{t-2} \left(F_2(n_i) + \frac{1}{2^{n_i+\dots+n_t}} \right) + F_2(n_{t-1}) + \frac{1}{2^{n_{t-1}+n_t}} + F_2(n_t) + \frac{1}{2^{n_t}} \\ &= \sum_{i=1}^{t-2} \left(F_2(n_i) + \frac{1}{2^{n_i+\dots+n_t}} \right) + F_2(n_{t-1} + n_t) \\ &\leq \sum_{i=1}^{t-2} \left(F_2(n_i) + \frac{1}{2^{n_i+\dots+n_t}} \right) + F_2(n_{t-1} + n_t) + \frac{1}{2^{n_{t-1}+n_t}} \\ &= \sum_{i=1}^{t-1} \left(F_2(n_i^*) + \frac{1}{2^{n_i^*+\dots+n_t^*}} \right), \end{aligned}$$

where

$$n_j^* = \begin{cases} n_j & \text{if } j = 1, \dots, t-2, \\ n_{t-1} + n_t & \text{if } j = t-1. \end{cases}$$

Now, the claimed upper bound in (iii) can be established again by a straightforward induction and is omitted. The remaining two recurrences are the same, up to a factor of 2 in a toll function, so let us consider the first one. Suppose that the root n is split as $n = n_1 + n_2$ and, inductively, that subsequent splittings of n_1 and n_2 follow the balanced power of 2 pattern. Assume without loss that $n_1 \geq n_2$ and let

$$n_1 = 2^{k_1} + 2^{k_2} + \dots + 2^{k_i} \quad \text{and} \quad n_2 = 2^{\ell_1} + 2^{\ell_2} + \dots + 2^{\ell_j},$$

be the respective binary representations (note that $n_1 \geq n_2$ implies $k_1 \geq \ell_1$). Thus, n_1 and n_2 are split as

$$n_1 = 2^k + m_1, \quad n_2 = 2^\ell + m_2,$$

where k is either $k_1 - 1$ or k_1 and ℓ is either ℓ_1 or $\ell_1 - 1$. We have

$$2^{k-1} \leq m_1 < 2^{k+1}, \quad 2^{\ell_1} \leq n_2 < 2^{\ell_1+1}.$$

The rest of the argument consists on considering various cases and reshuffling the nodes correspondingly. For example, if both m_1 and n_2 are less than 2^k we replace the node $n_1 = 2^k + m_1$ by 2^k and the node n_2 by $n_2 + m_1$ (subsequently split into n_2 and m_1). If all other splittings are kept the same, this operation will increase the value since the term $1/2^{2^k+m_1}$ will be replaced by a larger value $1/2^{n_2+m_1}$, and that could be further increased by applying the balanced power of 2 rule to the new node $n_2 + m_1$. Other cases are handled similarly, and we omit the rest of the details. \square

5.3. Expected value

In order to analyze a normalized distribution of $F(n)$ we will first find, asymptotically at least, its expected value and the variance. Given a composition of n let $m_k^{(n)}$ be the multiplicity of a part size k and let $\mu_k^{(n)}$ be the expected multiplicity of that size.

Let $f(n)$ and $v(n)$ denote the expected value and the variance of $F(n)$, respectively, and set $t(n) = ET(n)$. Grouping together the terms in recurrence relation (13) by their sizes, using linearity of expectation and the assumption that, given a composition, the distributions of $F(n_i)$ depend only on the size n_i , we have

$$\begin{aligned} f(n) &= EF(n) = E\left(T(n) + \sum_{i=1}^t F(n_i)\right) \\ &= t(n) + E\left(\sum_{j=1}^{n-1} \sum_{i=1}^t I_{n_i=j} F(n_i)\right) \\ &= t(n) + \sum_{j=1}^{n-1} E\left(\sum_{i=1}^t I_{n_i=j} F(n_i)\right) = t(n) + \sum_{j=1}^{n-1} \mu_j^{(n)} EF(j) \\ &= t(n) + \sum_{j=1}^{n-1} \mu_j^{(n)} f(j). \end{aligned}$$

Thus we obtain the following recurrence for $f(n)$:

$$f(n) = \sum_{j=1}^{n-1} \mu_j^{(n)} f(j) + t(n), \tag{16}$$

with $f(1)$ given (for WHT computations we set $f(1) = 0$). In order to proceed, we need some information about the quantities involved. Most of these have been studied for random compositions. Since in our model we disallow one trivial composition of n we need to adjust these results.

Lemma 3. *With the above notation, the following are true:*

- (i) $\mu_k^{(n)} = \frac{2^{n-1}}{2^{n-1} - 1} \frac{n + 3 - k}{2^{k+1}}$,
- (ii) $t_1(n) = \frac{(n + 1)2^{n-2} - 1}{2^n(2^{n-1} - 1)}$,
- (iii) $t_2(n) = \frac{1}{2} + \frac{1}{2^n}$.

Proof. Let κ_0 denote the trivial composition of n with one part. We denote the probability and the expectation over the set of all compositions by P_0 and E_0 , respectively. The relationship between E and E_0 is, of course, $E(\cdot) = E_0(\cdot | \kappa \neq \kappa_0)$. We begin with (iii); we need to find the expected value of

$$T_2(n) = \sum_{i=1}^t \frac{1}{2^{n_i+n_{i+1}+\dots+n_t}}.$$

To this end, first consider the expectation over all 2^{n-1} compositions of n . Denoting it by g_n and conditioning on the value of n_t , which is k with probability $1/2^k$, for $1 \leq k < n$, and n with probability $1/2^{n-1}$, we can write

$$\begin{aligned} g_n &= E_0 \frac{1}{2^{n_t}} \left(1 + \dots + \frac{1}{2^{n-n_t}} \right) \\ &= \sum_{k=1}^n P_0(n_t = k) E_0 \left(\frac{1}{2^{n_t}} \left(1 + \dots + \frac{1}{2^{n-n_t}} \right) \middle| n_t = k \right) \\ &= \frac{1}{2^{n-1}} \frac{1}{2^n} + \sum_{k=1}^{n-1} \frac{1}{2^k} E_0 \left(\frac{1}{2^k} \left(1 + \dots + \frac{1}{2^{n-k}} \right) \middle| n_t = k \right) \\ &= \frac{1}{2^{n-1}} \frac{1}{2^n} + \sum_{k=1}^{n-1} \frac{1}{2^k} \frac{1}{2^k} (1 + g_{n-k}) = \frac{1}{2^{2n-1}} + \sum_{k=1}^{n-1} \frac{1}{4^k} + \sum_{k=1}^{n-1} \frac{g_{n-k}}{4^k} \\ &= \frac{1}{3} + \frac{2}{3 \cdot 4^n} + \sum_{k=1}^{n-1} \frac{g_k}{4^{n-k}}, \end{aligned}$$

from which follows that $g_n = \frac{1}{2}$ is a solution. Discarding one composition amounts to computing the conditional expectation

$$\begin{aligned} E T_2(n) &= E_0(T_2(n) | \kappa \neq \kappa_0) = E_0(T_2(n) | n_t < n) = \frac{1}{P_0(n_t < n)} E_0 T_2(n) I_{n_t < n} \\ &= \frac{2^{n-1}}{2^{n-1} - 1} E_0 T_2(n) I_{n_t < n}. \end{aligned}$$

But

$$E_0 T_2(n) I_{n_t < n} = E_0 T_2(n) - E_0 T_2(n) I_{n_t = n} = \frac{1}{2} - \frac{1}{2^{n-1}} \frac{1}{2^n},$$

which proves (iii). As for the proofs of the first two assertions we use the facts that if all 2^{n-1} compositions of n are considered then the expected multiplicity of a part size k is $(n + 3 - k)/2^{k+1}$ (see, [19]) and that the number of parts t is equidistributed with $1 + \text{Bin}(n - 1, \frac{1}{2})$ random variable [13]. Thus, its expected value is $(n + 1)/2$. Since the one composition we disallow has one part of size n and multiplicity one and all other sizes have multiplicity zero, after adjusting in the same manner as above we obtain

$$\mu_k^{(n)} = \frac{2^{n-1}}{2^{n-1} - 1} \frac{n + 3 - k}{2^{k+1}},$$

which proves (i), and furthermore

$$\begin{aligned} \frac{n + 1}{2} &= E_0 t = E_0 t I(\kappa = \kappa_0) + E_0 t I(\kappa \neq \kappa_0) = \frac{1}{2^{n-1}} + P_0(\kappa \neq \kappa_0) E_0(t | \kappa \neq \kappa_0) \\ &= \frac{1}{2^{n-1}} + (1 - \frac{1}{2^{n-1}}) E t, \end{aligned}$$

from which (ii) follows. \square

Having computed the coefficients and tolls, we now turn to solving (16). This can be accomplished by elementary means. Rewriting

$$f(n) = \frac{2^{n-1}}{2^{n-1} - 1} \sum_{k=1}^{n-1} f(k) \frac{n + 3 - k}{2^{k+1}} + t(n)$$

as

$$\frac{2^{n-1} - 1}{2^{n-1}} f(n) = \sum_{k=1}^{n-1} f(k) \frac{n + 3 - k}{2^{k+1}} + t(n) \frac{2^{n-1} - 1}{2^{n-1}},$$

writing a similar expression replacing n by $n + 1$ and subtracting the former from the latter we obtain

$$\begin{aligned} & \frac{2^n - 1}{2^n} f(n + 1) - \frac{2^{n-1} - 1}{2^{n-1}} f(n) \\ &= \sum_{k=1}^n f(k) \frac{n + 4 - k}{2^{k+1}} - \sum_{k=1}^{n-1} f(k) \frac{n + 3 - k}{2^{k+1}} + t(n + 1) \frac{2^n - 1}{2^n} - t(n) \frac{2^{n-1} - 1}{2^{n-1}} \\ &= \frac{f(n)}{2^{n-1}} + \sum_{k=1}^{n-1} \frac{f(k)}{2^{k+1}} + t(n + 1) \frac{2^n - 1}{2^n} - t(n) \frac{2^{n-1} - 1}{2^{n-1}}, \end{aligned}$$

which yields

$$\frac{2^n - 1}{2^n} f(n + 1) = f(n) + \sum_{k=1}^{n-1} \frac{f(k)}{2^{k+1}} + t(n + 1) \frac{2^n - 1}{2^n} - t(n) \frac{2^{n-1} - 1}{2^{n-1}}.$$

Once again, writing a similar expression replacing $n + 1$ by n and subtracting we get

$$\begin{aligned} & \frac{2^n - 1}{2^n} f(n + 1) - \frac{2^{n-1} - 1}{2^{n-1}} f(n) \\ &= f(n) - f(n - 1) + \frac{f(n - 1)}{2^n} + t(n + 1) \frac{2^n - 1}{2^n} - 2t(n) \frac{2^{n-1} - 1}{2^{n-1}} + t(n - 1) \frac{2^{n-2} - 1}{2^{n-2}}, \end{aligned}$$

which, after solving for $f(n + 1)$, gives

$$f(n + 1) = 2f(n) - f(n - 1) + t(n + 1) - 4t(n) \frac{2^{n-1} - 1}{2^n - 1} + 4t(n - 1) \frac{2^{n-2} - 1}{2^n - 1}.$$

This can be easily solved for $f_{n+1} - f_n$ and after summation yields.

Theorem 4. *The solution of a recurrence (16) with the initial value $f(1)$ and toll function $t(n)$ is given by*

$$f(n) = nf(1) + \sum_{j=1}^{n-1} (n - j)\gamma_{j+1},$$

where $\gamma_2 = t(2)$, and for $j \geq 2$,

$$\gamma_{j+1} = t(j + 1) - \frac{4}{2^n - 1} \left((2^{j-1} - 1)t(j) - (2^{j-2} - 1)t(j - 1) \right).$$

In particular, for $i = 0, 1, 2$ there exist constants ϕ_i such that we have

$$f_i(n) \sim (f_i(1) + \phi_i)n.$$

Numerically, $\phi_0 = 0.073 \dots$, $\phi_1 = 0.152 \dots$, $\phi_2 = 0.271 \dots$.

5.4. Variances

Recurrences for variances may be derived in the same fashion as those for the expected values. Let $v(n) = \text{var}(F(n))$. The following elementary property of variance (easiest to find in texts on statistics, e.g., [5]) will be handy. If X is any random variable and \mathcal{A} a σ -algebra, then

$$\text{var}(X) = E \text{var}_{\mathcal{A}}(X) + \text{var}(E_{\mathcal{A}}(X)), \tag{17}$$

where $\text{var}_{\mathcal{A}}(X) = E((X - E(X|\mathcal{A}))^2|\mathcal{A})$ and $E_{\mathcal{A}}(X) = E(X|\mathcal{A})$ denote the conditional variance, and the conditional expectation given \mathcal{A} , respectively.

We will use the above with \mathcal{A} being a σ -algebra generated by compositions. That is to say that conditioning on \mathcal{A} means fixing a particular composition of n and if $\kappa = (n_1, \dots, n_t)$ was fixed, the conditional distribution of, say $F(n)$, is that of

$$\sum_{i=1}^t F(n_i) + T(n),$$

where we think of n_i 's, t , and $T(n)$ as deterministic (the last statement is certainly true in all three cases of our interest, and it is a reasonable assumption in general). Thus, the randomness is only in $F(n_1), \dots, F(n_t)$ and once n_i 's are fixed these are independent random variables. This, plus the fact that the variance is invariant under translations by a constant, and that, conditionally on \mathcal{A} , t is nonrandom, yields

$$\text{var}_{\mathcal{A}}(F(n)) = \text{var}_{\mathcal{A}}\left(\sum_{i=1}^t F(n_i) + T(n)\right) = \text{var}_{\mathcal{A}}\left(\sum_{i=1}^t F(n_i)\right) \tag{18}$$

$$= \sum_{i=1}^t \text{var}_{\mathcal{A}}(F(n_i)) = \sum_{j=1}^{n-1} m_j^{(n)} v(j), \tag{19}$$

where $m_j^{(n)}$ is the multiplicity of part size j . By the same reasoning, for the conditional expectation we get

$$\text{E}_{\mathcal{A}}(F(n)) = \text{E}_{\mathcal{A}}\left(\sum_{i=1}^t F(n_i) + T(n)\right) = \sum_{i=1}^t \text{E}F(n_i) + \text{E}_{\mathcal{A}}T(n) = \sum_{j=1}^{n-1} m_j^{(n)} f(j) + \text{E}_{\mathcal{A}}T(n). \tag{20}$$

It follows from (17), (19), and (20) that

$$v(n) = \text{E} \text{var}_{\mathcal{A}}(F(n)) + \text{var}(\text{E}_{\mathcal{A}}(F(n))) = \sum_{j=1}^{n-1} \mu_j^{(n)} v(j) + \text{var}\left(\sum_{j=1}^{n-1} m_j^{(n)} f(j) + \text{E}_{\mathcal{A}}T(n)\right), \tag{21}$$

which is exactly the same recurrence as (16) with toll function equal to $\text{var}(\sum_{j=1}^{n-1} m_j^{(n)} f(j) + \text{E}_{\mathcal{A}}T(n))$. In each of the three cases this term can be quite precisely computed, but does not appear to have a workable closed formula. For example, for $T_1(n)$, observing that the number of parts in a composition is the sum of all multiplicities, the toll $\text{var}(\sum_{j=1}^{n-1} m_j^{(n)} f(j) + t/2^n)$ can be written as

$$\text{var}\left(\sum_{j=1}^{n-1} m_j^{(n)} (f_1(j) + 1/2^n)\right).$$

However, solving the recurrences is more problematic since now we know the tolls only asymptotically and the recurrences are quite sensitive, since small values of n contribute significantly. Nonetheless, it is readily seen, that the tolls are linear functions of n (that is because the $m_j^{(n)}$ is asymptotically distributed like $\text{Bin}(\lceil n/2 \rceil, 1/2^j)$ and have asymptotically enough independence to show that covariances are negligible). Linearity of tolls implies that the solutions of the recurrences are also linear. Since in the next section we will provide another argument showing asymptotic linearity of the variances we will omit further details and will just state the result.

Proposition 5. *There exist absolute constants ω_0, ω_1 , and ω_2 such that for $i = 0, 1, 2$ we have*

$$v_i(n) \sim (v_i(1) + \omega_i)n.$$

Linearity of the variances is enough to establish convergence in distribution of $F(n)$ to normal random variable.

5.5. Limiting distribution

We will show in this section that the random variables $F(n)$, normalized to have mean zero and the variance 1, converge in distribution to a standard normal random variable (we refer the reader to [3] for all necessary background from probability theory that will be used throughout the remainder of this section). That is,

Theorem 6. *For $k = 0, 1, 2$ we have*

$$\frac{F_k(n) - f_k(n)}{\sqrt{v_k(n)}} \implies N(0, 1),$$

where $N(0, 1)$ denotes the normal random variable with mean zero and variance 1.

Proof. This is a consequence of basic properties of random compositions and a central limit theorem for martingales. We will rely on a representation of a random composition of n given in (14). Restricting attention to compositions with at least two parts amounts to considering $\Gamma_k I(\Gamma_k < n)$'s rather than Γ_k 's and since the probability that these two are different is exponentially small and inconsequential from the point of view of the limiting law, from now on we will consider *all* compositions of n . In that case, τ defined by Eq. (15) is distributed like a $1 + \text{Bin}(n - 1, \frac{1}{2})$ random variable, and thus is tightly concentrated around its expected value which is $(n + 1)/2$. In particular, as is well known

$$\frac{\tau}{E\tau} = \frac{2\tau}{n + 1} \longrightarrow 1, \tag{22}$$

in probability as $n \rightarrow \infty$. Again, let us consider a generic quantity

$$\frac{F(n) - f(n)}{\sqrt{v(n)}} = \frac{\sum_{i=1}^t F(n_i) - E \sum_{i=1}^t F(n_i)}{\sqrt{v(n)}} + \frac{T(n) - t(n)}{\sqrt{v(n)}}.$$

Since $v(n)$ is of order n and all three tolls are bounded, $(T(n) - t(n))/\sqrt{v(n)}$ goes to zero and since the limiting distribution is continuous, this term can be neglected. The quantity whose limiting distribution we want to study in the present set-up, is

$$\tilde{W}_n = \sum_{k=1}^{\tau-1} F(\Gamma_k) + F\left(n - \sum_{j=1}^{\tau-1} \Gamma_j\right),$$

where for an integer valued, positive random variable Z , $F(Z)$ is a random variable, whose conditional distribution given $Z = k$ is the distribution of $F(k)$, and the random variables $F(\Gamma_j)$, $j \geq 1$ are independent. Since, as follows from computations carried out in [13], $n - \sum_{j=1}^{\tau-1} \Gamma_j$ does not differ much from Γ_τ , (the difference is bounded in expectation), and $F(\Gamma)$ grows linearly with Γ , we may replace \tilde{W}_n by $W_{\tau \wedge n} = \sum_{j=1}^{\tau \wedge n} F(\Gamma_j)$; more specifically, we have

$$\frac{\tilde{W}_n - W_{\tau \wedge n}}{\sqrt{v(n)}} \longrightarrow 0,$$

in probability as $n \rightarrow \infty$. Thus, it suffices to consider a sequence $\{W_{\tau \wedge n} : n \geq 1\}$ and we want to show that

$$\frac{W_{\tau \wedge n} - E W_{\tau \wedge n}}{\sqrt{v(n)}} = \frac{\sum_{k=1}^n I(\tau \geq k) F(\Gamma_k) - \sum_{k=1}^n E I(\tau \geq k) F(\Gamma_k)}{\sqrt{v(n)}},$$

converges in distribution to a standard normal random variable. The plan is to apply the martingale central limit theorem [3, Theorem 35.12]. Let \mathcal{F}_n , $n \geq 0$ be an increasing sequence of σ -fields with $\mathcal{F}_0 = \{\emptyset, \Omega\}$. There is a canonical way of turning any integrable random variable into a martingale, by taking successive conditional expectations. We will apply this procedure to random variables

$$W_{\tau \wedge n} - E W_{\tau \wedge n}, \quad n \geq 1.$$

Specifically, for $k \geq 1$ we set

$$\mathcal{F}_k = \sigma(\Gamma_1, \dots, \Gamma_k, F(\Gamma_1), \dots, F(\Gamma_k)),$$

and for $n \geq 1$, $\mathcal{F}_{n,k} = \mathcal{F}_k$. We now set

$$X_{n,k} := E(W_{\tau \wedge n} | \mathcal{F}_{n,k}), \quad n \geq 1, k = 0, 1, \dots, n,$$

and

$$Y_{n,k} := X_{n,k} - X_{n,k-1}, \quad n \geq 1, k = 1, \dots, n.$$

Then, for $n \geq 1$

$$W_{\tau \wedge n} - E W_{\tau \wedge n} = \sum_{k=1}^n Y_{n,k},$$

and $(Y_{n,k})$ is a triangular array of martingale difference sequences, just as required for an application of [3, Theorem 35.12]. (Strictly speaking we should have used $(W_{\tau \wedge n} - \mathbb{E}W_{\tau \wedge n})/\sqrt{v(n)}$, so that σ in that theorem is 1, but this is just a matter of normalization, and for the sake of notational convenience we will denote by $X_{n,k}$ and $Y_{n,k}$ the quantities before normalization.) In this notation, conditions (35.35) and (35.36) of that theorem become, respectively,

$$\frac{\sum_{k=1}^n \mathbb{E}(Y_{n,k}^2 | \mathcal{F}_{k-1})}{v(n)} \rightarrow 1, \tag{23}$$

in probability, and

$$\frac{\sum_{k=1}^n \mathbb{E}Y_{n,k}^2 I(|Y_{n,k}| \geq \varepsilon v(n))}{v(n)} \rightarrow 0, \tag{24}$$

for each $\varepsilon > 0$. Writing $\mathbb{E}_m(\cdot)$ for $\mathbb{E}(\cdot | \mathcal{F}_m)$ we have

$$\begin{aligned} X_{n,k} &= \mathbb{E}_k \left(\sum_{j=1}^n I(\tau \geq j) F(\Gamma_j) \right) \\ &= \sum_{j=1}^k I(\tau \geq j) F(\Gamma_j) + \mathbb{E}_k \left(\sum_{j=k+1}^n I(\tau \geq j) F(\Gamma_j) \right) \\ &= \sum_{j=1}^k I(\tau \geq j) F(\Gamma_j) + \mathbb{E}_k \left(\sum_{j=k+1}^n I(\tau \geq j) \mathbb{E}_{j-1} F(\Gamma_j) \right) \\ &= \sum_{j=1}^k I(\tau \geq j) F(\Gamma_j) + \mathbb{E}_k \left(\sum_{j=k+1}^n I(\tau \geq j) \mathbb{E} f(\Gamma_j) \right) \\ &= \sum_{j=1}^k I(\tau \geq j) F(\Gamma_j) + \mathbb{E} f(\Gamma) \cdot \mathbb{E}_k \left(\sum_{j=k+1}^n I(\tau \geq j) \right), \end{aligned}$$

where we have used the fact that both Γ_j and $F(\Gamma_j)$ are independent of \mathcal{F}_{j-1} and thus the conditional expectation $\mathbb{E}_{j-1} F(\Gamma_j)$ is equal to

$$\mathbb{E} F(\Gamma_j) = \mathbb{E} \mathbb{E}(F(\Gamma_j) | \Gamma_j) = \mathbb{E} f(\Gamma_j) = \mathbb{E} f(\Gamma),$$

where Γ is a random variable equidistributed with Γ_j . Hence, the differences $Y_{n,k}$ are

$$Y_{n,k} = X_{n,k} - X_{n,k-1} = I(\tau \geq k) F(\Gamma_k) + \mathbb{E} f(\Gamma) \left(\mathbb{E}_k \sum_{j=k+1}^n I(\tau \geq j) - \mathbb{E}_{k-1} \sum_{j=k}^n I(\tau \geq j) \right).$$

Now, the conditional distribution of $\sum_{j=k+1}^n I(\tau \geq j)$ given \mathcal{F}_k is the number of parts following the first k parts $\Gamma_1, \dots, \Gamma_k$. This is the number of parts in a randomly chosen composition of $n - S_k$ where $S_k = \Gamma_1 + \dots + \Gamma_k$. Thus

$$\mathcal{L} \left(\sum_{j=k+1}^n I(\tau \geq j) \middle| \mathcal{F}_k \right) = 1 + \text{Bin} \left(n - 1 - S_k, \frac{1}{2} \right),$$

provided $S_k \leq n - 1$, i.e., $\tau \geq k$. In particular, for every $k \geq 0$,

$$\mathbb{E}_k \sum_{j=k+1}^n I(\tau \geq j) = \frac{n + 1 - S_k}{2} I(\tau \geq k + 1).$$

Using that and then $I(\tau \geq k + 1) = I(\tau \geq k) - I(\tau = k)$ we obtain

$$\begin{aligned} Y_{n,k} &= I(\tau \geq k)F(\Gamma_k) + \mathbb{E}f(\Gamma) \left(\frac{n+1-S_k}{2} I(\tau \geq k+1) - \frac{n+1-S_{k-1}}{2} I(\tau \geq k) \right) \\ &= I(\tau \geq k)F(\Gamma_k) + \mathbb{E}f(\Gamma) \left(\frac{-\Gamma_k}{2} I(\tau \geq k) - \frac{n+1-S_k}{2} I(\tau = k) \right) \\ &= I(\tau \geq k) \left(F(\Gamma_k) - \frac{\Gamma_k}{2} \mathbb{E}f(\Gamma) \right) + \frac{\mathbb{E}f(\Gamma)}{2} (S_k - n - 1) I(\tau = k) \\ &:= d_{n,k} + e_{n,k}. \end{aligned}$$

Each of the two terms is a martingale difference, and we will show the total contribution to the sum coming from $e_{n,k}$'s is negligible. Writing \Pr_{k-1} for the conditional probability given \mathcal{F}_{k-1} we have

$$\begin{aligned} \Pr_{k-1}(S_k - n - 1 = m, \tau = k) &= \Pr_{k-1}(\Gamma_k + S_{k-1} - n - 1 = m, S_{k-1} \leq n, S_k \geq n) \\ &= I(S_{k-1} < n) \Pr_{k-1}(\Gamma_k + S_{k-1} - n - 1 = m, \Gamma_k \geq n - S_{k-1}) \\ &= I(\tau \geq k) \Pr_{k-1}(\Gamma_k + 1 - (n - S_{k-1}) - 2 = m, \Gamma_k \geq n - S_{k-1}) \\ &= I(\tau \geq k) \Pr(\Gamma + 1 - (n - S_{k-1}) - 2 = m, \Gamma \geq n - S_{k-1}), \end{aligned}$$

where Γ is a $\text{GEOM}(\frac{1}{2})$ random variable and by independence of Γ_k and \mathcal{F}_{k-1} , in the last line S_{k-1} is considered fixed, and \Pr applies to Γ only. Furthermore, by the memoryless property of Γ (see, [5, Section 3]), conditionally on $\Gamma \geq \ell \geq 1$, $\Gamma + 1 - \ell$ is equidistributed with Γ . Hence, the last probability above is equal to

$$\Pr(\Gamma \geq n - S_{k-1}) \Pr(\Gamma + 1 - (n - S_{k-1}) - 2 = m | \Gamma \geq n - S_{k-1}) = 2^{S_{k-1}+1-n} \Pr(\Gamma - 2 = m).$$

The first three moments of $\Gamma - 2$ are 0 and 2, and 6 which translates into $\mathbb{E}_{k-1} e_{n,k} = 0$ (confirming that $e_{n,k}$ is a martingale difference),

$$\mathbb{E}_{k-1} e_{n,k}^2 = \frac{\mathbb{E}^2 f(\Gamma)}{4} I(\tau \geq k) 2^{S_{k-1}+1-n} \cdot 2 = I(\tau \geq k) 2^{S_{k-1}-n} \mathbb{E}^2 f(\Gamma), \tag{25}$$

and

$$\mathbb{E}_{k-1} |e_{n,k}|^3 = 3 I(\tau \geq k) 2^{S_{k-1}-1-n} \mathbb{E}^3 f(\Gamma). \tag{26}$$

Hence, we immediately obtain

$$\sum_{k \geq 1} \mathbb{E}_{k-1} e_{n,k}^2 = O(1) \cdot \sum_{k=1}^{\tau} 2^{-k} \leq O(1) \cdot \sum_{k=1}^{\infty} 2^{-k} = O(1). \tag{27}$$

This, in turn implies that

$$|\mathbb{E}_{k-1} Y_{n,k}^2 - \mathbb{E}_{k-1} d_{n,k}^2| \leq \mathbb{E}_{k-1} e_{n,k}^2 + 2 \mathbb{E}_{k-1} |d_{n,k} e_{n,k}| \leq \mathbb{E}_{k-1} e_{n,k}^2 + 2 (\mathbb{E}_{k-1} d_{n,k}^2)^{1/2} (\mathbb{E}_{k-1} e_{n,k}^2)^{1/2},$$

where in the last step we used the conditional version of Cauchy–Schwartz inequality. Summing up yields

$$\left| \sum_{k=1}^n \mathbb{E}_{k-1} Y_{n,k}^2 - \sum_{k=1}^n \mathbb{E}_{k-1} d_{n,k}^2 \right| \leq \sum_{k=1}^n \mathbb{E}_{k-1} e_{n,k}^2 + 2 \left(\max_{1 \leq j \leq n} \mathbb{E}_{j-1} d_{n,j}^2 \right)^{1/2} \sum_{k=1}^n (\mathbb{E}_{k-1} e_{n,k}^2)^{1/2} = O(1),$$

since $\mathbb{E}_{j-1} d_{n,j}^2 = O(1)$ (uniformly in j) and, by the same argument as for (27),

$$\sum_{k=1}^n (\mathbb{E}_{k-1} e_{n,k}^2)^{1/2} = O(1),$$

we infer that

$$\left| \sum_{k=1}^n \mathbb{E}_{k-1} Y_{n,k}^2 - \sum_{k=1}^n \mathbb{E}_{k-1} d_{n,k}^2 \right| = O(1). \tag{28}$$

Now

$$E_{k-1}d_{n,k}^2 = I(\tau \geq k)E_{k-1}(F(\Gamma_k) - \frac{\Gamma_k}{2} Ef(\Gamma))^2 = I(\tau \geq k)E(F(\Gamma) - \frac{\Gamma}{2} Ef(\Gamma))^2, \tag{29}$$

and

$$\sum_{k=1}^n EY_{n,k}^2 = v(n). \tag{30}$$

Hence we get

$$\frac{\sum_{k=1}^n E_{k-1}Y_{n,k}^2}{v(n)} = \frac{\sum_{k=1}^n E_{k-1}d_{n,k}^2 + O(1)}{\sum_{k=1}^n Ed_{n,k}^2 + O(1)} = \frac{\tau}{E\tau} + O(1/n),$$

which in view of (22) implies (23). To prove (24) we just write

$$\begin{aligned} EY_{n,k}^2 I(|Y_{n,k}| \geq \varepsilon\sqrt{v(n)}) &\leq E \frac{|Y_{n,k}|^3}{\varepsilon\sqrt{v(n)}} I(|Y_{n,k}| \geq \varepsilon\sqrt{v(n)}) \leq \frac{c}{\sqrt{v(n)}} E(|d_{n,k}|^3 + |e_{n,k}|^3) \\ &= O(1/\sqrt{v(n)}), \end{aligned}$$

since both $d_{n,k}$ and $e_{n,k}$ have uniformly bounded third moments (for $d_{n,k}$'s this is clear, and for $e_{n,k}$'s follows from (26)). Hence

$$\sum_{k=1}^n EY_{n,k}^2 I(|Y_{n,k}| \geq \varepsilon\sqrt{v(n)}) = O(n/\sqrt{v(n)}) = O(\sqrt{n}),$$

which implies (24) and completes the proof. \square

Remark. Note that (28)–(30) imply that $v(n) \sim wn$, where $w = E(F(\Gamma) - (\Gamma/2)Ef(\Gamma))^2/2$. While we did use the linearity of the variance at the beginning of the proof of Theorem 6 we only needed a superlinearity of $v(n)$, which is evident from recurrence (16).

6. Comparison with binary splits

It may be of some interest to compare the situation with one in which, at every stage, only random binary splits are allowed. This leads to a quicksort type of recurrence. Such recurrences have been thoroughly analyzed in a series of papers, culminating in [15], which gives the most complete picture up to date. Since our toll functions fall into “small toll function” category, the limiting distribution is normal, so one only has to find asymptotic mean and the variance. But this can be readily done. For example, following the usual steps, we obtain that for the binary splits the expected value $f_0(n)$ satisfies

$$\frac{f_0(n)}{n} = \frac{f_0(1)}{1} + \sum_{j=2}^n \frac{1}{j2^j} - \sum_{j=2}^n \frac{j-2}{(j-1)j2^{j-1}} = f_0(1) + \frac{1}{n2^n} + 2 \sum_{j=2}^{n-1} \frac{1}{j(j+1)2^j},$$

which, writing $1/(j(j+1))$ as $1/j - 1/(j+1)$ and using the fact that

$$\sum_{j=1}^{\infty} \frac{1}{j2^j} = \ln 2$$

yields

$$f_0(n) = n \left(f_0(1) + \frac{3}{2} - 2 \ln 2 \right) + O(1/2^n).$$

Also, for binary splits we have

$$t_1(n) = \frac{1}{2^{n-1}} \quad \text{and} \quad t_2(n) = \frac{1}{n-1} + \frac{1}{2^n} \left(1 - \frac{2}{n-1} \right),$$

which gives

$$f_1(n) = n(f_1(1) + 3 - 4 \ln 2) + O(1/2^n)$$

and

$$f_2(n) = n \left(f_2(1) + \frac{5}{2} - 3 \ln 2 \right) + O(1/2^n).$$

Numerically, the coefficients in front of linear terms are: $f_0(1) + 0.113705\dots$, $f_1(1) + 0.227411\dots$, and $f_2(1) + 0.4205584\dots$. To compare with the corresponding values for all compositions with at least two parts, see Theorem 4.

Acknowledgments

We would like to thank an anonymous referee for several suggestions that led to improvements in the presentation of our results.

References

- [1] V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, C. Tan (Eds.), Computational science—ICCS 2001, Lecture Notes in Computer Science, Vol. 2073, Springer, 2001, (Session on architecture-specific automatic performance tuning).
- [2] K.G. Beauchamp, Applications of Walsh and Related Functions, Academic Press, New York, 1984.
- [3] P. Billingsley, Probability and Measure, third ed., Wiley, New York, 1995.
- [4] J. Bilmes, K. Asanović, C.W. Chin, J. Demmel, Optimizing matrix multiply using PhiPAC: a portable, high-performance, ANSI C coding methodology, in: Proc. Supercomputing, ACM SIGARC, 1997. (<http://www.icsi.berkeley.edu/~bilmes/phipac>).
- [5] G. Casella, R.L. Berger, Statistical Inference, Wadsworth & Brooks/Cole, Belmont, CA, 1990.
- [6] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, C. Whaley, K. Yelick, Self adapting linear algebra algorithms and software, Proc. IEEE 93 (2) (2005) (special issue on program generation, optimization, and adaptation).
- [7] D.F. Elliott, K.R. Rao, Fast Transforms: Algorithms, Analyses, Applications, Academic Press, New York, 1982.
- [8] J.A. Fill, S. Janson, Smoothness and decay properties of the limiting quicksort density function, in: D. Gardy, A. Mokkadem (Eds.), Mathematics and computer science: algorithms, trees, combinatorics and probabilities, Trends in Mathematics, Birkhäuser Verlag, London, 2000, pp. 53–64.
- [9] P. Flajolet, G. Gonnet, C. Puech, J.M. Robson, Analytic variations on quad trees, Algorithmica 10 (1993) 473–500.
- [10] P. Flajolet, R. Sedgewick, Analytic Combinatorics, zeroth ed., third printing, 2005. (<http://algo.inria.fr/flajolet/Publications/AnaCombi1to9.pdf>).
- [11] M. Frigo, S.G. Johnson, The design and implementation of FFTW3, Proc. IEEE 93 (2) (2005) (special issue on program generation, optimization, and adaptation).
- [12] P. Hitczenko, G. Louchard, Distinctness of compositions of an integer: a probabilistic analysis, Random Struct. Alg. 19 (2001) 407–437.
- [13] P. Hitczenko, C.D. Savage, On the multiplicity of parts in a random composition of a large integer, SIAM J. Discrete Math. 18 (2004) 418–435.
- [14] H.-J. Huang, Performance analysis of an adaptive algorithm for the Walsh–Hadamard transform, Master’s Thesis, Drexel University, 2002.
- [15] H.-K. Hwang, R. Neininger, Phase change of limit laws in the quicksort recurrence under varying toll functions, SIAM J. Comput. 31 (2002) 1687–1722.
- [16] H.-K. Hwang, T-S. Tsai, An asymptotic theory for recurrence relations based on minimalization and maximization, Theoret. Comput. Sci. 290 (2003) 1475–1501.
- [17] J.R. Johnson, M. Püschel, In search for the optimal Walsh–Hadamard transform, Proc. ICASSP, Vol. 4, 2000, pp. 3347–3350.
- [18] J.R. Johnson, R.W. Johnson, D. Rodriguez, R. Tolimieri, A methodology for designing, modifying, and implementing Fourier transform algorithms on various architectures, Circuits Systems Signal Process. 9 (4) (1990) 449–500.
- [19] B. Kheifets, The expected value and the variance of a multiplicity of a given part size in a random composition of an integer, J. Combin. Math. Combin. Comput. 52 (2005) 65–68.
- [20] F.J. MacWilliams, N.J. Sloane, The Theory of Error-Correcting Codes, North-Holland, Amsterdam, 1992.
- [21] H. Mahmoud, Sorting: A Distribution Theory, Wiley, New York, 2000.
- [22] D. Mirković, S.L. Johnsson, Automatic performance tuning in the UHFFT library, in: Proc. ICCS, Lecture Notes in Computer Science, Vol. 2073, Springer, Berlin, 2001, pp. 71–80.
- [23] M. Püschel, J.M.F. Moura, J.R. Johnson, D. Padua, M. Veloso, B.W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R.W. Johnson, N. Rizzolo, SPIRAL: code generation for DSP transforms, Proc. IEEE 93 (2) (2005) (special issue on program generation, optimization, and adaptation).
- [24] M. Régnier, A limiting distribution for quicksort, RAIRO Theoret. Inform. Appl. 23 (1989) 335–343.
- [25] U. Röslér, A limit theorem for quicksort, RAIRO Theoret. Inform. Appl. 25 (1991) 85–100.
- [26] S. Roura, An improved master theorem for divide and conquer recurrences, J. ACM 48 (2001) 170–205.
- [27] C. Van Loan, Computational frameworks for the fast Fourier transform, Frontiers in Applied Mathematics, Vol. 10, Society for Industrial and Applied Mathematics, Philadelphia, 1992.
- [28] R.C. Whaley, J. Dongarra, Automatically tuned linear algebra software (ATLAS), in: Proc. Supercomput. 1998. (<http://math-atlas.sourceforge.net/>).