CLARKSON UNIVERSITY

COMBINATORIAL PROPERTIES OF HEAPSORT

A Dissertation

by

Ahmad Al-Jaber

Department of Mathematics and Computer Science

Submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Mathematics

November 1984

Accepted by the Graduate School

November 19, 1984 Richard J. Arenz

Date Dean

ACKNOWLEDGEMENTS

The author wishes to thank his advisor Dr. Ernst E. Doberkat for his guidance, support and encouragement during this project. The many discussions between Dr. Doberkat and myself have been invaluable and are gratefully acknowledged. I wish to thank Drs. Eytan Barouch, Susan Conry, Mark Goldberg and James Lynch, for their interest as well as for their valuable time. Cindy Martin must be thanked for her typing (and retyping); Faye Gates is thanked for her editing skills. Last but not least, to my family, for the support, faith, confidence and patience - thank you.

TABLE OF CONTENT

ABSTRACT

Several aspects related to the combinatorial properties of heapsort are discussed in this thesis. A recursion formula for the number of heaps satisfying a given condition between any two offsprings with the same parent is given and several properties of heaps are discussed including a new algorithm to generate the set of all heaps of any size. Also in this work we define second order trees which have a great importance in the study of the complexity of Williams' algorithms to generate a heap. We discuss this kind of trees and we prove that the generating function of the number of trees satisfies a nonlinear differential difference equation. The numerical computation and the asymptotic expansion for a quantity related to this nonlinear differential difference equation is given in this work. Finally, we give an upper bound for the number of the second order trees generated from the set of all heaps of size N where N has the form 2^k-1 for any positive integer k.

0. INTRODUCTION

A heap is a data structure proposed to sort a sequence of elements; it can also be used to implement priority queues (a priority queue, as defined in Aho, Hopcroft and Ullman, is a data structure that supports finding the maximum or minimum element from a given ordered set efficiently [AHU - p. 110]). Formally, a N-heap can be described as an array x[1..N] of real numbers with

$$x[j] > x[\lfloor j/2 \rfloor] \quad \text{for } 2 \leq j \leq N.$$

Two important operations are allowed on a heap. Insertion, the first one, allows a new element to be inserted into the heap. Thus one can create a heap from a set by repeated insertions of the set elements into an initially empty heap. The second operation allows the smallest (or largest) element to be deleted from the heap. In particular, the smallest/largest element can be found efficiently.

Heaps have been useful in several areas such as sorting [KNU2,AHU] and operating systems [HAB] (specifically in scheduling). For example, we have a set of jobs waiting for service from a processor and every job is assigned some priority. The basic situation is that one has to decide which job should get service next; usually one picks the next job from a structure that allows to make this decision based on the job's priority. There are several possibilities for the implementation of such a structure. One of the most efficient in this case is a heap. Heaps are also useful in finding the minimum spanning tree and the shortest path in a given graph [AHU- p. 110]. Considerable research has been done in this area; Floyd [FLO] gave his algorithm for generating a N-heap from a given permutation of size N as follows:

Input: array $\bar{x}$[1..N]

Output: x[1..N] organized as a heap.

```
Method:  0.  Procedure heapify (k);
             If k is not a leaf and if a son of k has a smaller label
             than k, then let j be the son with the smallest label;
             interchange x[k] with x[j];
                heapify(j);
                End{If};
             End{heapify}
         1.  For k := ⌊N/2⌋ downto 1 do
                heapify(k);
             End. {Algorithm}
```

Floyd's algorithm has an attractive randomness property [KNU2]. Williams [WIL] gives another algorithm to generate a N-heap from a given permutation of size N, but this algorithm does not satisfy the randomness property. (A full description of Williams' algorithm is given in Section 0.2.2). In [PSI], Porter and Simon analyzed the average number of interchanges generated from inserting a random element into a random N-heap; using Williams' algorithm, in other words, they investigated the expected number of interchanges required to generate an (N+1)-heap by inserting the (N+1)-element into the given N-heap. They found that this average is bounded by a constant of about 1.61. This analysis was refined and extended in [DOB1] by Doberkat. He derived formulas for the expected numbers of comparisons and the higher moment generated from inserting a random element into a random heap and gave the asymptotic expansion for these formulas. In his paper [DOB4] Doberkat analyzed Floyd's algorithm and gave formulas for the expected number of interchanges and comparisons required to generate a heap of size N. All analyses are based on Knuth [KNU2] who derives a number of combinatorial characteristics for Floyd's algorithm such as the number of interchanges and some relation-

ships between the left offspring and right offspring. Knuth [KNU2] also derives a formula for the number of heaps generated from the set of all permutations of size N and proves this number to be $\frac{N!}{\prod_{i=1}^{N} S_{i,N}}$, where $S_{i,N}$ is the size of the subtree rooted at node i, (see 0.2.1 for the tree representation).

A brief description of the results obtained in this thesis are in order now. We will transform the analysis of Williams' algorithm into a combinatorial problem by defining a new type of tree which we call a second order tree (SOT). This type of tree is the main concept behind the analysis of William's algorithm. (For further details about second order trees, see Chapter II, Section II.1). From this transformation several questions arise concerning the properties of SOT, viz., the combinatorial characterization of these trees, the cardinality of the set of the second order trees generated from the set of all heaps of size N and the weight of the given tree generated from a heap of size N. We will show in Chapter II, Section II.2 that this type of tree cannot be completely characterized by the number of nodes, leaf or edges, the height of the tree or by other simple combinatorial properties. This will be demonstrated by two different examples in Chapter II, Section II.2. While one is a second order tree and the other is not, both trees do have the same number of nodes, leaves, edges and the same height.

One of the main results we prove in Chapter II is Theorem II.3.5 which gives us the number of the second order trees generated from the set of all heaps of siz In II.4 we use the exponential generating function to prove that the exponential gene ing function of the number of second order tree satisfies a non-linear differentia difference equation. Several properties of the second order tree are given in Chapter II, including coefficients for some terms, asymptotic expansions and

the number of the second order trees according to the number of offsprings (maximum, minimum). Also in Chapter II we present some numberical computation for the number of the SOT. The main result in Chapter III is the setting of an upper bound for the SOT, it is given in Corollary III.1.2.

In Chapter I of this work we prove some combinatorial properties of heaps. The principal result of this chapter is Theorem I.2 , where we give a new algorithm to generate the set of all heaps of size N from a subset of the heaps of size (N-1).

0.1 Definitions

0.1.1 - DEF: Let $\{1,\ldots,N\}$ be represented as a binary tree as follows: 1 represents the root, and for any node i, $2 \leq i \leq N, \lfloor i/2 \rfloor$ is the parent of node i. If i is even, then i is the left offspring of $\lfloor i/2 \rfloor$; if i is odd, then i is the right offspring of $\lfloor i/2 \rfloor$. Given an array x[1..N] of real numbers, label each node i by x(i). This labeling constitutes a heap iff every offspring has a greater label than its parent.

0.1.2 - Williams' algorithm [WIL].

In a very simple way, Williams' algorithm is nothing but a straight insertion sort in each path in the complete binary tree representation of a given permutation. The algorithm can be written as follows:

```
Input: x[1..N] as a permutation
Output: x[1..N] organized as a heap
Method: x(0) = -∞
For i :=1 to N do
Begin
   p := i;  q := ⌊i/2⌋;  x := x(i);
   While x(q) > x do
      Begin
         x(p) := x(q);  p := q;  q := ⌊p/2⌋;
      End; {While}
   x(p) := x;
End  {For}
End. {Algorithm}
```

0.1.3 - DEF: Let x[1..N] be a heap of size N and define

$$t(N,0) := N;$$

$$t(N,i+1) := \left\lfloor \frac{t(N,i)}{2} \right\rfloor \quad \text{for } 0 \leq i \leq \lfloor \log_2 N \rfloor - 1.$$

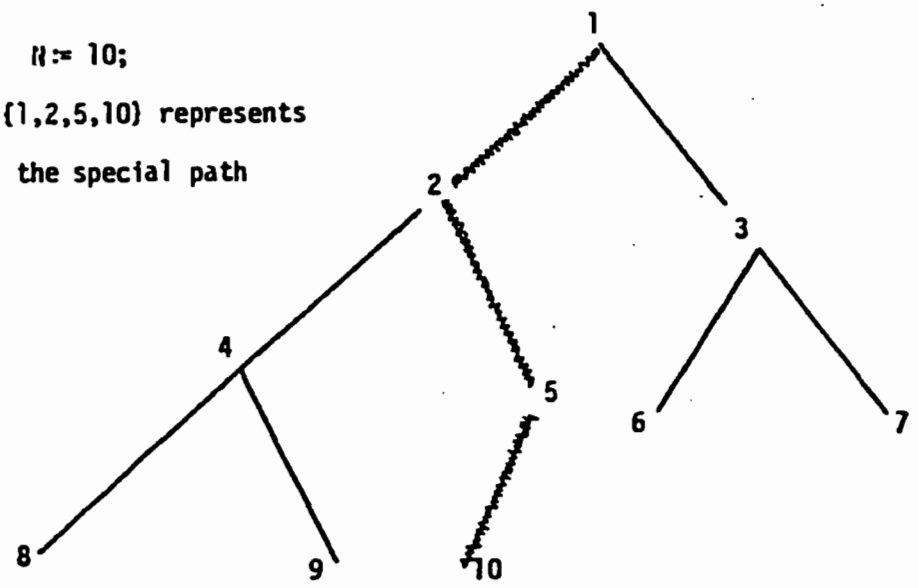
Then

$$(t(N,i);\ 0 \leq i \leq \lfloor \log_2 N \rfloor)$$

is called the special path and $t(N,i)$, for $0 \leq i \leq \lfloor \log_2 N \rfloor$, is called a special node (Knuth [KNU2] - Section 5.2.3).

Example: - $N := 10$;

$(1,2,5,10)$ represents the special path



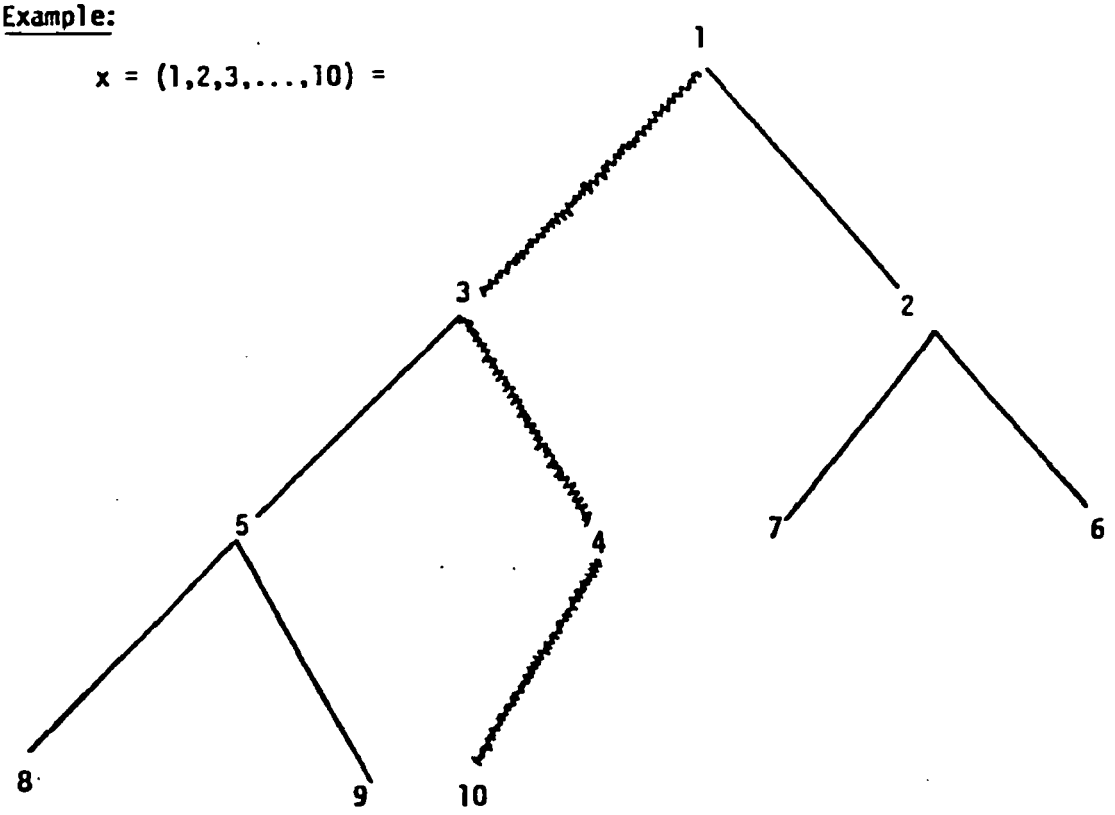
0.1.4 - DEF:

Define $b(N,i)$ to be the brother of $t(N,i)$ for $0 \leq i \leq \lfloor \log_2 N \rfloor$. If N is odd, $b(N,0)$ is defined too; otherwise, let $b(N,0) = \infty$. To create a heap of size N from a heap of size $N-1$, the set of all possible interchanges after inserting the key of the node N depends on the relation between special nodes and their brothers. Given a heap $x[1..N]$, we define the skewness of x (i.e., the set of all possible interchanges from inserting the key of N) by

$$\text{skewness}(x) := \max\{r; x(t(N,i)) < x(b(N,i)) \text{ for all } i,\ 0 \leq i < r\}.$$

Example:

x = (1,2,3,...,10) =



Skewness of x = 2, since $x(10) < \infty$, $x(5) < x(4)$ and $x(2) > x(3)$.

0.2 Generating functions

One of the important subjects in the area of the analysis of algorithms is finding a recursion formula which reflects the mathematical solution to a given problem. Several techniques have been proposed to solve recursion formula One of these techniques uses generating function, see [GKu, KNU2, LIU, LOU, McB, MOO]. There is more than one type of generating function; for example, ordinary generating function, exponential generating f tion, Lambert series,... . For convenience we will define the exponential and the ordinary generating function, respectively.

0.2.1 - DEF:

Let $\{a_n\}_{n\geq 0}$ be a sequence of real numbers. Then

a) $\sum_{n\geq 0} \frac{a_n}{n!} t^n$

is called the exponential generating function of $\{a_n\}_{n\geq 0}$.

b) $\sum_{n\geq 0} a_n t^n$

is called the ordinary generating function of $\{a_n\}_{n\geq 0}$.

0.3 Asymptotics

0.3.1 DEF: Suppose the function f has a singularity at α. The singularity is called algebraic if f(z) can be written as analytic function near α plus a finite sum of terms of the form

$$(1 - z/\alpha)^{-w} g(z) \qquad (0.3.1)$$

where

g is a function which is analytic near α and

w is a complex number not equal to 0, -1, -2,...

0.3.2 - Theorem:(Darboux; Greene and Knuth [GKu]):

Suppose $A(z) = \sum_{n \geq 0} a_n z^n$ is analytic near 0 and has only algebraic singularities on its circle of convergence. Let w be the maximum of the weights at these singularities. Denote by α_k, w_k and g_k the values of α, w and g for those terms of the form (0.3.1) of weight w. Then

$$a_n - \frac{1}{n} \sum_k \frac{g_k(w_k) n^{w_k}}{\Gamma(w_k)\alpha_k^n} = o(r^{-n} n^{w-1}), \quad \text{as } n \to \infty$$

where $r = |\alpha_k|$, is the radius of convergence of A(z), and $\Gamma(s)$ is the gamma function.

0.3.3 - Theorem:(Stirling formula [BEN, OLV[):

$$x! = (2\pi x)^{1/2} (\frac{x}{e})^x (1 + O(\frac{1}{x})) .$$

For real x as $x \to \infty$.

CHAPTER I

I.0 Introduction

In this chapter we will study some of the properties of heap sort, viz, the number of heaps with maximum and minimum skewness. We will develop a ne recursion formula which gives us the number of heaps satisfying a certain order relation between two offsprings and, finally, we will give a new algorithm to generate the set of all heaps of size N from a proper subset of all heaps of size N-1 .

One of the major differences between our algorithm and the two known algorithms by Floyd and Williams is that we are using a proper subset of the set of all heaps of size N-1 to generate the set of all heaps of size N. In contrast, in the other two algorithms one has to use at least the set of all heaps of size N-1 to generate the set of all heaps of size N. The advantage to our algorithm is that clearly a great amount of time and space can be saved in comparison with the two known algorithms.

I.1 Properties of heap

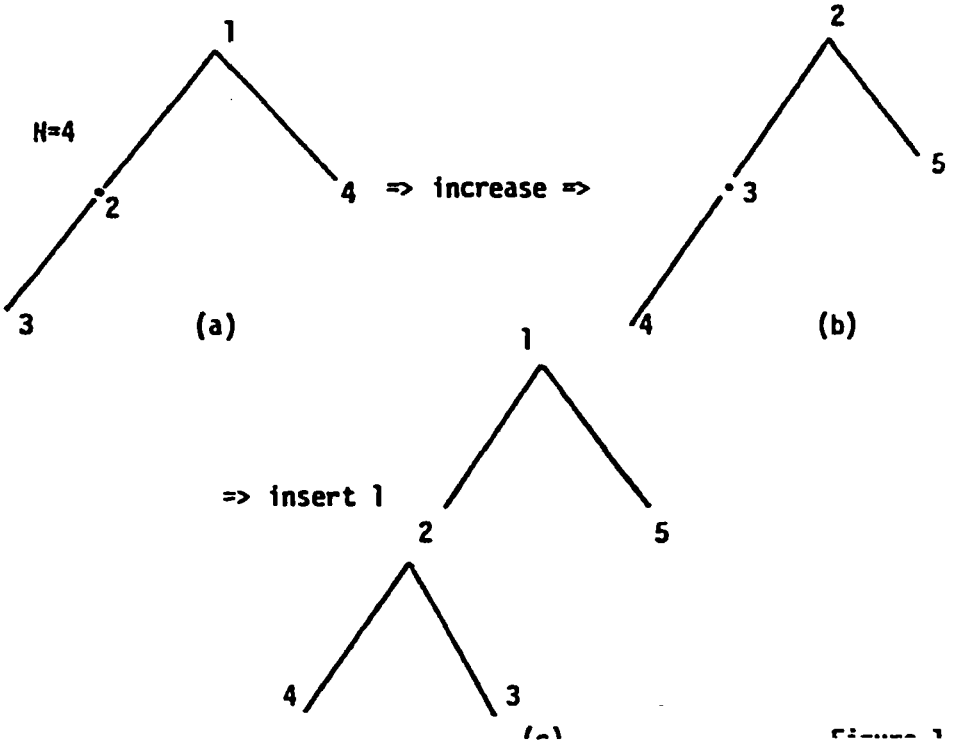
Let A_N be the set of all permutations over $\{1,\ldots,N\}$. Define H_N over A_N as

$H_N := \{x;\ x \in A_N,$ and x has the heap property$\}$

Then we have the following properties:

I.1.1 - Lemma: Let $H_{N+1}^{(max)} := \{x^{(max)};\ x^{(max)}$ is a heap of size $(N+1)$ with maximal skewness$\}$ then there exists a one-to-one correspondence between H_N and $H_{N+1}^{(max)}$.

Proof: "$\geq$" - Let $x \in H_N$; increase each label in x by 1. Then every label in x is greater than 1; insert 1, we get a new heap $\hat{x}$ of size $N+1$. $\hat{x}$ has maximal skewness since the labels on the special path in $\hat{x}$ are produced by increasing the label of the same path in x by 1 and the labels on that path are sifted down one level by inserting 1 as in fig. 1.



"$\leq$" If $x \in H_{N+1}^{(max)}$, deleting the root always gives us a heap of size N; hence this inequality follows. □

For the sequel we will fix N as the size of the heap, M is the size of x_ℓ and N-M-1 is the size of x_R, and $\nu(y)$ is the number of heaps of size y.

I.1.2 - Lemma: Let $x \in H_N$ and $x_\ell(x_R)$ be the left (right) subheap of x, respectively, as in Fig. 2. Then the number of heaps that satisfy

$$x(2) > x(3) \qquad (*)$$

is

$$\nu(M) \cdot \nu(N-M-1)\binom{N-2}{M} .$$

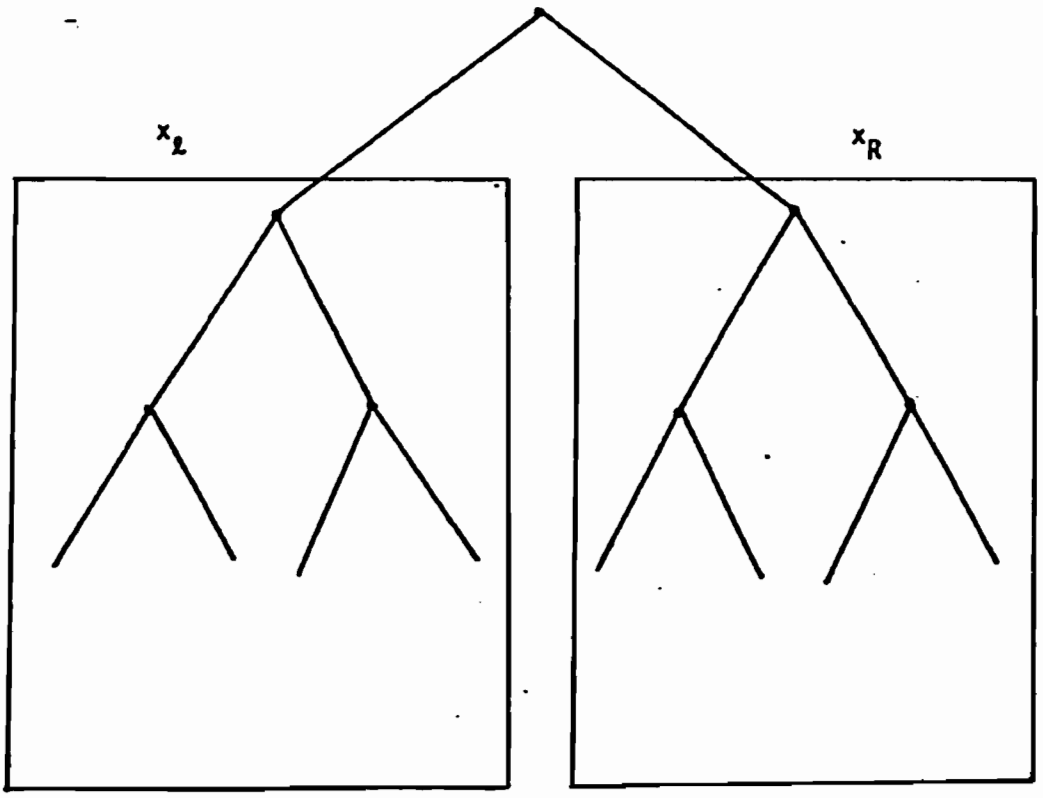


Figure 2

Proof: - If $x(2) > x(3)$, we must have $x(3) = 2$ and $x(1) = 1$. These two elements are fixed, and so we have $\binom{N-2}{M}$ possible choices for the elements of the left subheap and the right subheap; and, for each choice we have $\nu(M)$ left subheaps and $\nu(N-M-1)$ right subheaps, therefore, we have

$$\binom{N-2}{M}\nu(M)\nu(N-M-1)$$

heaps satisfying condition (*). □

Lemma I.1.2 can be generalized to any two nodes i, i + 1 with the same parent. This is done in the next lemma.

I.1.3- Lemma: Let $x \in H_N$, let i be a node in x such that $i \neq 2$ and both i and i+1 have the same parent; assume $\nu'(M)$ is the number of heaps of size M where j has the same position in the binary tree corresponding to $(1,\ldots,M)$ as i the given tree as in fig. 3. Then the number of heap with $x(i) > x(i+1)$ is

$$\binom{N-1}{M}\nu'(M)\nu(N-M-1).$$

Example: -

If i = 4, hence i + 1 = 5,

let $x(4) < x(5)$ in x which is equivalent to $x_\ell(2) < x_\ell(3)$

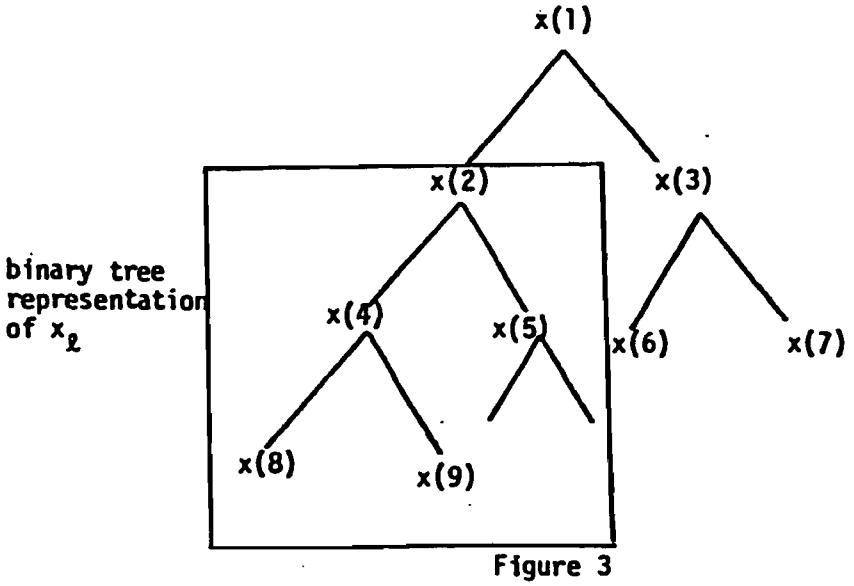


Figure 3

Proof of Lemma I.1.3 - The proof for this lemma follows by induction from lemma I.1.2. □

These two lemmas (I.1.2 - I.1.3) can be used to study the properties of heaps and in particular to investigate the relation between the elements of a heap.

One can use these lemmas to give a very easy proof for Knuth's formula

$$|H_{1N}| = \frac{N!}{\prod_{i=1}^{N} S_{i,N}}$$

for the size of H_{1N}. Here $S_{i,N}$ is the size of the subtree of $(1,\ldots,N)$ rooted at node i.

I.2 An algorithm to generate the set of all heaps

In order to verify the algorithm given below we have to provide two simple lemmas.

I.2.1 - Lemma: Let N,j be a natural number which satisfy the following conditions:

i) $j \leq N - 1$;

ii) j is even and $2j > N$.

Hence, j is a leaf and a left offspring in the tree representation.

Let B be some predicate over H_N such that B is independent of $x(j) < x(j+1)$ and define the following sets:

$${}_{<}H_N^B := \{x \in H_N;\ B(x) \text{ holds and } x(j) < x(j+1)\};$$

$${}_{>}H_N^B := \{x \in H_N;\ B(x) \text{ holds and } x(j) > x(j+1)\};$$

$$H_N^B := \{x \in H_N;\ B(x) \text{ holds}\}, \text{ then}$$

$$|{}_{<}H_N^B| = |{}_{>}H_N^B| = \frac{1}{2}\,|H_N^B| \ .$$

Proof:

It is clear that

$$|{}_{<}H_N^B| + |{}_{>}H_N^B| = |H_N^B| \quad \text{since } {}_{<}H_N^B \cap {}_{>}H_N^B = \phi.$$

We have to show that there exists a one-to-one correspondence between ${}_{<}H_N^B$ and ${}_{>}H_N^B$. This may be easily seen by interchanging x(j) and x(j+1). □

I.2.2-DEF : Fix N and define $m := m(N) := \{j;\ j$ satisfies condition i) and ii) in Lemma I.2.1$\}$.

I.2.3-Lemma: Let $\phi \neq m' \subseteq m$ and let $B_{m'}$ be some predicate over H_N such that $B_{m'}$ is independent of $x(j) < x(j+1)$ for all $j \in m'$. Define

$$ {}_{<}H_N^{B_{m'}} := \{x \in H_N;\ B_{m'}(x) \text{ holds and } x(j) < x(j+1) \text{ for all } j \in m'\}; $$

$$ {}_{>}H_N^{B_{m'}} := \{x \in H_N;\ B_{m'}(x) \text{ holds and } x(j) > x(j+1) \text{ for all } j \in m'\}; $$

$$ H_N^{B_{m'}} := \{x \in H_N;\ B_{m'}(x) \text{ holds}\} \text{ then} $$

$$ \left|{}_{<}H_N^{B_{m'}}\right| = \frac{1}{2^{|m'|}} \left|H_N^{B_m}\right| . $$

Proof: by induction

$|m'| = 1$ by Lemma I.2.1.

Assume the statement is correct for all $m'' \subsetneq m'$. Let $m' = m'' \cup \{j\}$ with $j \in m'$ and $j \notin m''$; thus

$$ |m'| = |m''| + 1. $$

But with

$$ {}_{<}H_N^{B_{m'}} = \{x \in {}_{<}H_N^{B_{m''}};\ B_{m''}(x) \text{ holds and } x(j) < x(j+1)\} \text{ this implies } \left|{}_{<}H_N^{B_{m'}}\right| = \frac{1}{2}\left|{}_{<}H_N^{B_{m''}}\right|, $$

hence the conclusion follows. □

Using lemma I.2.3 we can construct the following algorithm to generate the set of all heaps of size n by making use of some subset of H_{n-1} .

I.2.4 - Algorithm:

Input : heap of size N-1.

Output : sequence of heaps of size N.

Method : 0. define

$$\ell := \begin{cases} \frac{N}{2} & \text{if } N \text{ is even} \\ N-1 & \text{if } N \text{ is odd.} \end{cases}$$

{ℓ is the counter index}.

$$S'(N-1) := \begin{cases} m(N-1) & \text{if } N \text{ is odd or} \\ & \quad N \text{ is even with } \ell \notin m \\ m(N-1)-\ell, & \text{if } \ell \in m(N-1) \text{ and } N \text{ is even} \end{cases}$$

{m(N-1) is defined in I.2.2}.

1. For each $x \in {}_{<}H_N^{B_{S'}}$ do the following

(*) For I := x(ℓ) + 1 to N do

If I = N put x(N) = I

Else

increase each label in x greater than or equal to I by 1 and put x(N) := I;

End {If};

End {For};

(**) 2. For any $y \in {}_{<}H_N^{B_{S''}}$ generate all possible heaps by making use of all possible interchanges between y(i) and y(i+1) for all $i \in S''$.

where

$$= \begin{cases} s'(N), & \text{if } N \text{ is even} \\ s'(N-1) \cup \{N-1\}, & \text{if } N \text{ is odd} \end{cases}$$

Remarks:

(*) This loop generates $N - x(2)$ heaps; each heap is of size N and belongs to $<H_N^{B_{S''}}$.

(**) This step is executed $2^{|S''|}$ times for each $y \in <H_N^{B_{S''}}$.

1.2.4.1- Proof of the correctness of the algorithm:

In order to prove the correctness of the algorithm we have to show that each element in H_N is generated at least once. But in our proof we will prove every heap of size N is uniquely generated.

a) Assume that there exists an $x \in H_N$ which is generated twice in $<H_N^{B_{s''}}$ this implies there exists $x^{(1)}, x^{(2)} \in <H_{N-1}^{B_{s'}}$ such that $x^{(1)}, x^{(2)}$ generate x and $x^{(1)}$ is different from $x^{(2)}$. Thus the counters of $x^{(1)}$ and $x^{(2)}$ are equal, hence there exists i, $1 \leq i \leq N$, such that $x^{(1)}(i) \neq x^{(2)}(i)$. But by step 1 we have to increase each element greater than or equal to I by one; hence, $x(i) \neq x(i)$. This is a contradiction.

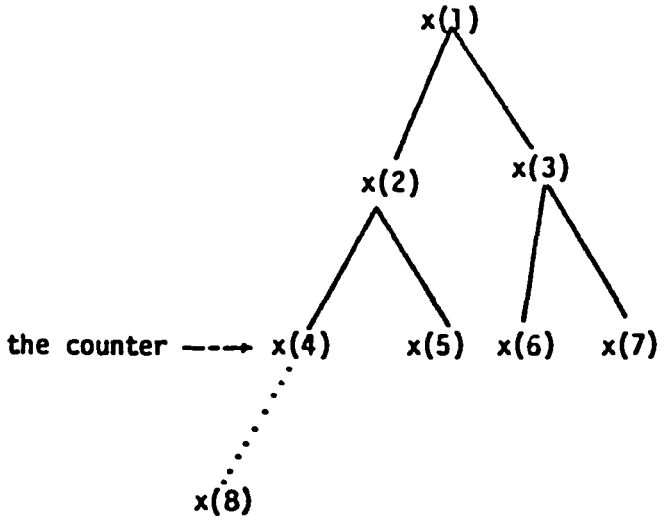
b) In order to prove that every $x \in H_N$ is generated by our algorithm — it is sufficient to show $<H_N^{B_{s''}} = <H_N^{B_{m(N)}}$ since every heap in H_N can be transformed into another heap in $<H_N^{B_{m(N)}}$ by I.2.2 . This can be done in the following way: for every $x \in H_N$ and for every $i \in m(N)$ interchange x(i) and x(i+1) if $x(i) < x(i+1)$. It is sufficient to show $m(N) = s''$

$$m(N) := \begin{cases} m(N-1) \cup \{N-1\}, & \text{if N is odd} \\ m(N-1), & \text{if } \lfloor \log_2 N-1 \rfloor = \lfloor \log_2 N \rfloor \text{ and N is even} \\ m(N-1) - \{\frac{N}{2}\}, & \text{otherwise} \end{cases}$$

$$= \begin{cases} s' \cup \{N-1\} & \text{if N is odd} \\ s', & \text{if N is even} \end{cases}$$

$= s''$. □

I.2.5 - Example: Let N-1=7, ℓ = 4, m(N-1) ={4,6}, s' ={6}, s" ={6}.



I.2.6- Example: Let N-1 =9, ℓ = 5, m(N-1)={6,8}, s' = {6,8},
s" = {6,8}, m(N) = {6,8} .

I.2.7- Example: N-1 =10, ℓ= 10, m(N-1) ={6,8}, s' = {6,8},
s" = {6,8,10} , m(N) = {6,8,10}.

To generate the set of all heaps of size 6 by algorithm 1.2.4 we will consider the following examples:

1.2.8- Example: N = 6.

0. ℓ = 3. s'(5) = {4}, s" = {4}

1. $H_5^{B_{s'}}$ = {12435, 12345, 12534, 13245}

$x^{(1)}$ = 12435 this implies from (*) we get

{124365, 124356}. .... (a)

$x^{(2)}$ = 12345 this implies from (*) we get

{123564, 123465, 123456}. ... (b)

$x^{(3)}$ = 12534 this implies from (*) we get

{125346}. ... (c)

$x^{(4)}$ = 13245 this implies from (*) we get

{142563, 143564, 132465, 132456}. ... (d)

this implies

$$_{<}H_6^{B_{s''}} = \{124365, 124356, 123564, 123465, 123456, 125346, 142563, 132564, 132465, 132456\}.$$

2. From every element in $_{<}H_6^{B_{s''}}$ we can generate two different heaps, for example, if y = 124365 we will have by (**)

124365 and 124635

This implies $|H_6^{B_{s''}}| = 20$ which is the number of heaps of size 6.

1.2.9 - Example: N = 7.

0. $\ell = 6$

$s'(6) = \{4\}$, $s'' = \{4,6\}$

$_{<}H_6^{B_{s'}}$ is the same as $_{<}H_6^{B_{s''}}$ in the previous example.

By (*) if x = 125346 we will get y = 1253467 $\in$ $_{<}H_7^{B_{s''}}$ and by (**) we get from y the following heaps

1253467, 1254367, 1253476 and 1254376.

We can repeat the same process for every y $\in$ $_{<}H_6^{B_{s'}}$ we will get $H_7^{B_{s'}}$.

CHAPTER II

II.0 Introduction

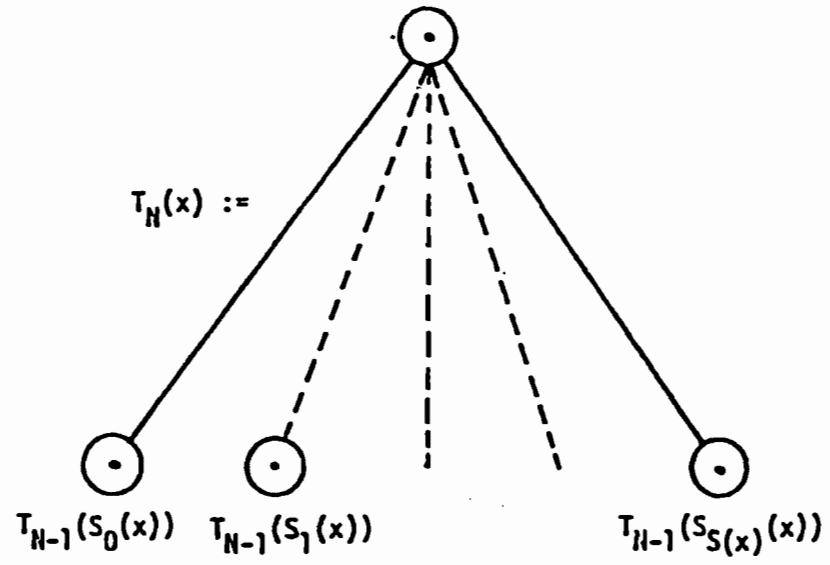
In this chapter we will study several topics related to the second order tree, in particular the definition of the second order tree and the cardinality of some classes of second order trees. Also we will give an asymptotic expansion for a quantity related to them; solutions for some terms and numerical computation are also provided in this chapter.

II.1 Combinatorial formulation of Williams algorithm

In order to study some aspects of the expected number of interchanges in Williams' algorithm (0.2.2), we will reformulate the problem in a combinatorial manner, but before that we have to provide some definitions and explanations.

Given $x \in H_N$, define the skewness of x by $S(x)$ and for every j, $0 \le j \le S(x)$; define the inverse $y := S_j(x)$ of x by $y(N) := x(t(N,j))$ and $y(t(N,i)) = x(t(N,i-1))$, $1 \le i \le j$; hence, $\{S_j(x);\ 0 \le j \le S(x)\}$ is exactly the set of all inverse images of x under the insertion process and in $S_j(x)$ the insertion will be exactly at node $t(N,j)$, requiring exactly j interchanges.

Now define for given $x \in H_N$ the tree $T_N(x)$ corresponding to x recursively by $T_1(x) := \theta$ (single node in case $x \in H_1$) and



The root of the subtree $T(S_j(x))$ is labeled j (from left to right).

From now on let us call the tree $T_N(x)$ the second order tree (SOT)

associated with x.

Define

$$J_{ii} := \{T_{ii}(x);\ \ x \in H_{ii}\} .$$

J_{ii} is the set of all second order tree corresponding to H.
Let

$$\ell(T_{ii}(x)) := \text{ the number of leaves for } T_{ii}(x)$$

and

$$L(J_{ii}) := \sum_{T_{ii} \in J_{ii}} \ell(T_{ii});$$

hence, $L(J_{ii})$ is the number of leaves for all second order trees corresponding to H.

Let $w(T_{ii})$ be the sum of the labels of $T_{ii} \in J_{ii}$, counting the root as 0 and

$$W(J_{ii}) := \sum_{T_{ii} \in J_{ii}} w(T_{ii}).$$

Hence $W(J_{ii})$ is the sum of all labels of all second order tree.

II.1.2 - Claim: Let $\mu(T) := |\{x \in H_{ii},\ T_{ii}(x) = T\}|$ be the multiplicity of $T \in J_{ii}$,

then

$$\sum_{T \in J_{ii}} \mu(T)\ell(T) = N!$$

Proof : "$\geq$" is clear since every permutation can be transformed into a heap.

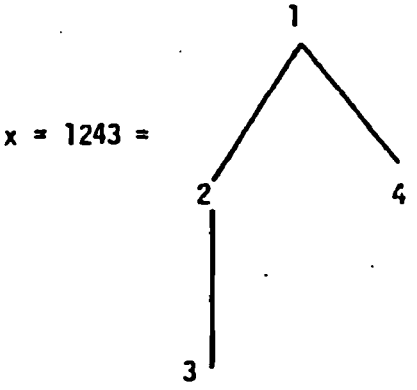
"$\leq$" $\mu(T)\ell(T)$ is the number of permutation of $\{1,\ldots,N\}$ that generate the tree T. □

II.1.3 - Claim : The expected number of interchanges is

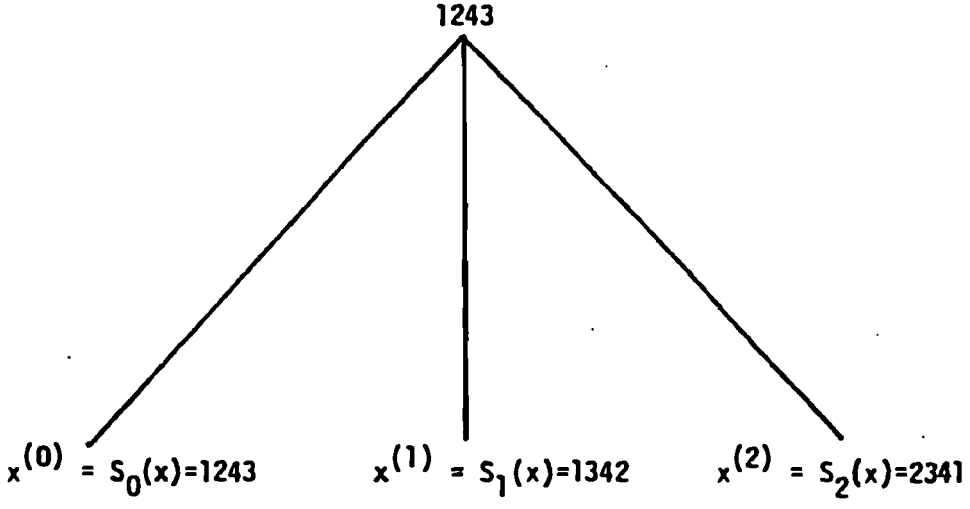
$$\frac{W(J_n)}{n!} .$$

Proof : The number of interchanges for any permutation is the sum of the labels of the tree associated to that permutation, i.e., $w(T_{ij})$. Thus the total number of interchanges is $\sum_{T_{ij} \in J_n} w(T_{ij}) = W(J_n)$. □

II.1.4 - Example :



The skewness of x is 2 since $3 = x(t(4,0)) < x(b(4,0)) = \infty$ and $2 = x(t(4,1)) < x(b(4,1)) = 4$. Thus we have the following inverse images



$x^{(0)}, x^{(1)}, x^{(2)} \in H_3$, but the skewness of $x^{(0)}, x^{(1)}, x^{(2)}$ is 0, thus we have in each case exactly one inverse image

$$S_0(x^{(0)}) = 1243 = y^{(0)}$$

$$S_0(x^{(1)}) = 1342 = y^{(1)}$$

$$S_0(x^{(2)}) = 2341 = y^{(2)} .$$

The skewness of $y^{(0)}, y^{(1)}$ and $y^{(2)}$ is 1; thus, we have two inverse images in each case

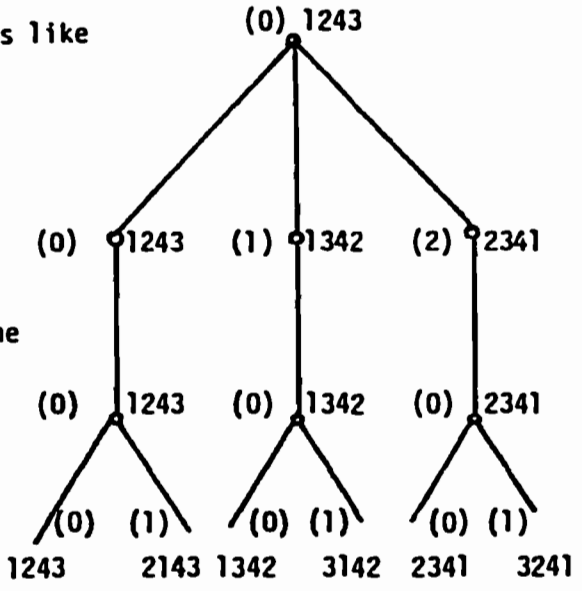
$$S_0(y^{(0)}) = 1243 = Z_0^{(0)}, \quad S_1(y^{(0)}) = 2143 = Z_0^{(1)}$$

$$S_0(y^{(1)}) = 1342 = Z_1^{(0)}, \quad S_1(y^{(1)}) = 3142 = Z_1^{(1)}$$

and

$$S_0(y^{(2)}) = 2341 = Z_2^{(0)}, \quad S_1(y^{(2)}) = 3241 = Z_2^{(1)} .$$

Therefore T_4 looks like



(o) represents the labels

Fig. II-1

II.1.5 - Problems : From the above explanation, the following questions arise concerning second order trees:-

a) Given a rooted tree T_n of height n-1, derive necessary and sufficient conditions for $T_n \in J_n$.

b) What is the cardinality of J_n?

c) How many trees have a given weight?

II.2 Necessary and sufficient conditions for the second order tree

From the definition of the second order tree it is immediate that every second order tree satisfies the following conditions :

(1) Every node in the second lowest level is binary since the skewness of any heap of size 2 is 1;

(2) All the leaves appear on the same level;

(3) For any second order tree T of height h-1 there is a heap x of size n such that T is generated by x.

Actually (1) and (2) are special cases of (3), but to make a quick decision one can check to see if (1) and (2) are satisfied; if they are we have to check condition (3), otherwise the tree is not a second order tree.

Unfortunately the second order tree cannot be identified by the number of edges and the number of nodes. In order to show that we will give the following two examples.

II.2.1 - Example :

T :=

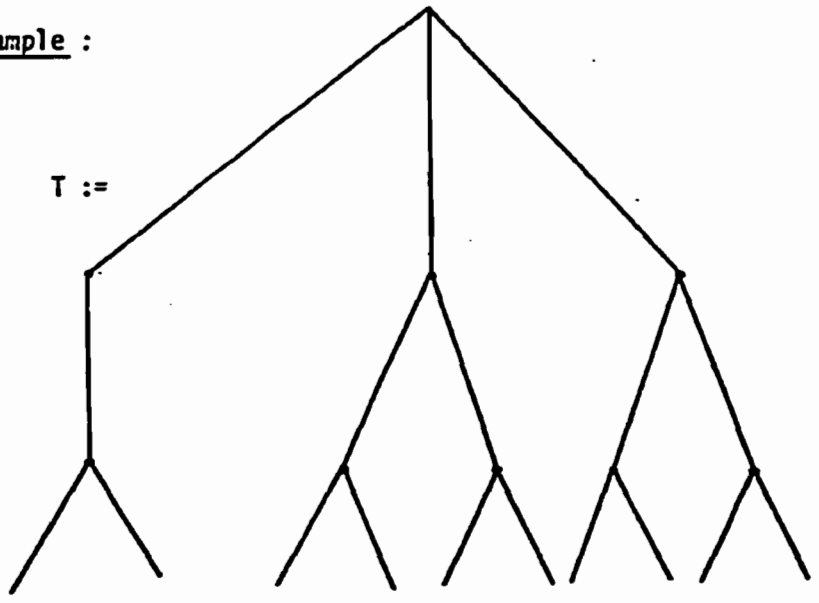


Fig. II-2

In this example the tree T is a second order tree generated by 1234 .

II.2.2 - Example :

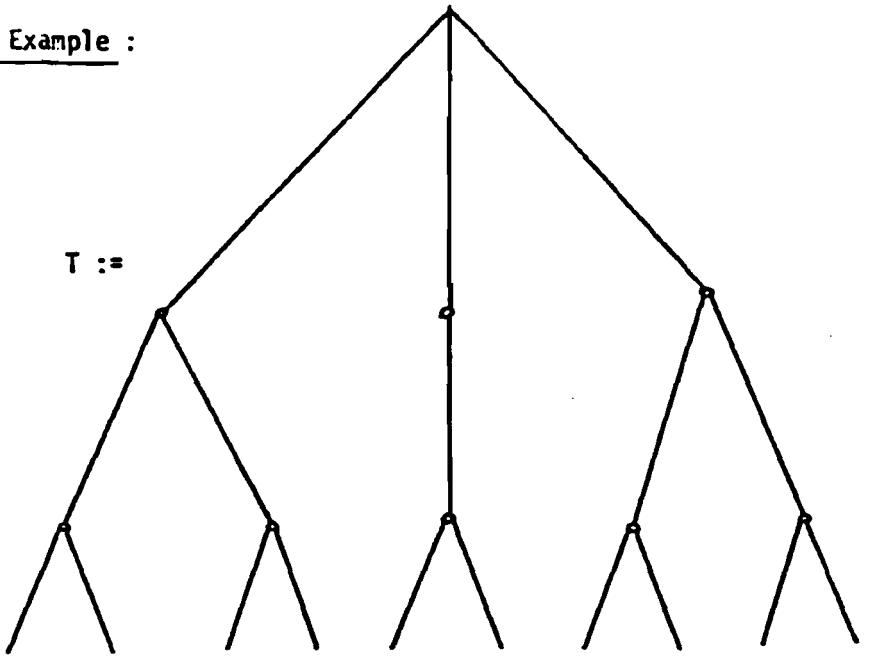


Fig. II-3

This tree T is not a second order tree since J_4 contains only three second order trees and T is not one of them.

One can easily see that the trees in (II.2.2) and (II.2.1) have the same height, the same number of nodes and the same number of edges.

Given a tree T to check whether or not T is a second order tree, assume the existence of a heap of size N if the height of T is N-1, and compare the given tree with the heap conditions (i.e., for any node i, $x(i) > x(\lfloor i/2 \rfloor)$). If it matches it is a second order tree; otherwise it is not.

By using [LOT], prop. 11.2.2, p. 217, $T_{N-1}(S_0(x)),\ldots,T_{N-1}(S_{S(x)}(x))$ are uniquely determining T_N. We can construct the following recursive procedure to check if T is a second order tree or not.

Input: $T_N(x)$ tree of height N-1

Output: $T_N(x)$ is a second order tree or not

{The file F is a global file containing the heap conditions initially and each time we call the procedure check we either have to terminate the algorithm or we have to add the new conditions to F}.

Method: 0. F := [heap condition for each i:

label (i) > label ($\lfloor i/2 \rfloor$), $2 \leq i \leq N$

and label (1) = 1];

Procedure check ($T_N(x)$)

Begin

m := the number of offspring of T_N;

For i := 0 to m-1 do

Begin

set label (t(N,i)) < label (b(N,i))

{condition (1) is the relation between an element of the special path and its brother}

Compare (1) with the contents of F

If it is compatible add condition (1) to F

Else

terminate the algorithm and write a message that this tree is not a second order tree.

End; {If}

End; {For}

For j := 0 to m-1 do calculate $S_j(x)$

For j := 0 to m-1 do check ($T_{N-1}(S_j(x))$)

End {Procedure};

End {Algorithm}.

II.2.3- Example :- Let T =  ; thus, assume there exists a heap

x(1)x(2)x(3) of size 3. This implies that the content of F initially is x(1) < x(2) and x(1) < x(3) or m=2. This implies x(3) < x(2), but x(3) < x(2). Compatible with the constant of F then, F becomes

$$F = \boxed{\begin{array}{l} x(1) < x(2) \\ x(1) < x(3) \end{array}}$$

$$F = \boxed{\begin{array}{l} x(1) < x(2) \\ x(1) < x(3) \\ x(3) < x(2) \end{array}} \quad ;$$

this implies $S_0(x) = x(1)x(2)$, $S_1(x) = x(3)x(2)$.

Now we have to check $T_2(S_0(x))$ and $T_2(S_1(x))$. Since they are binary they are compatible with F, and T is a second order tree. □

II.3 Counting second order trees

II.3.1 - DEF: Let $J_N^{(1)}$ be the set of all second order trees in which the root has only one offspring.

II.3.2 - Lemma:

$$|J_{N+1}^{(1)}| = |J_N| \quad \text{in case N is even.}$$

Proof:

"$\leq$" Given a tree $T_N \in J_N$ we know that there exists a heap $x \in H_N$ such that x generates T_N, but N+1 is greater than any element in x. Thus by insertion of N+1 into x we get a heap x' of size N+1 and N+1 becomes a leaf in $x' \in H_{N+1}$. This implies that the second order tree generated by x' has fanout one since N+1 is greater than x(N).

"$\geq$" Given a tree $T_{N+1}^{(1)} \in J_{N+1}^{(1)}$, hence there exists a heap $x \in H_{N+1}$ such that x generates $T_{N+1}^{(1)}$ and the skewness of x is zero. Now remove the element x(N+1) from x. We get a heap x' of size N, and since x(N+1) is a leaf and x(N+1) > x(N), (N is even), this operation will give us a heap of size N. But the second order tree generated by x' is the only subtree of $T_{N+1}^{(1)}$. This implies for any $T_{N+1}^{(1)}$ there exists T_N. □

II.3.3 - Lemma: Let $J_N^{(max)} := \{T_N^{(max)}; T_N^{(max)}$ is second order tree with maximal fanout at the root$\}$. Then

$$|J_{N+1}^{(max)}| = |J_N| .$$

(The proof for this Lemma is given in the next section on page 41).

Given $x \in H_{ii}$ with associated second order tree $T_{ii}(x)$, let $T_{ii}(x)$ have as labels for its nodes all permutations of size N that generate x.

Suppose the labels on level i are $x_{i,1},\ldots,x_{i,k}$ [thus $x_{i,j}$ is a permutation of $\{1,\ldots,N\}$ such that $x_{i,j}(1),\ldots,x_{i,j}(N-i)$ forms a heap and after i iterations, $x_{i,j}$ will become x under Williams' algorithm for $1 \leq j \leq k$].

Define

$$Z_{m_i} := \{x_{i,j}(N-i) : 1 \leq j \leq k\}$$

and

$$\hat{Z}_{m_i} := Z_{m_{(N-i-1)}} .$$

Thus $\hat{Z}_{m_i}$ can be described recursively as

(a) $x(i+1) \in \hat{Z}_{m_i}$

(b) $\hat{Z}_{m_{2(i+1)-1}} \subseteq \hat{Z}_{m_i}$

and

(c) If for $h \in \hat{Z}_{m_{2(i+1)}}$ there exists $y \in \hat{Z}_{m_{2(i+1)-1}}$ such that $h < y$ then $h \in \hat{Z}_{m_i}$.

From $\hat{Z}_{m_i}$ define the following sequence

$$y_{1:m_i} := y_1 \cdots y_{m_i}$$

as

$$y_1 := \min \hat{Z}_{m_i} \quad \text{then } y_1 = x(i+1),$$

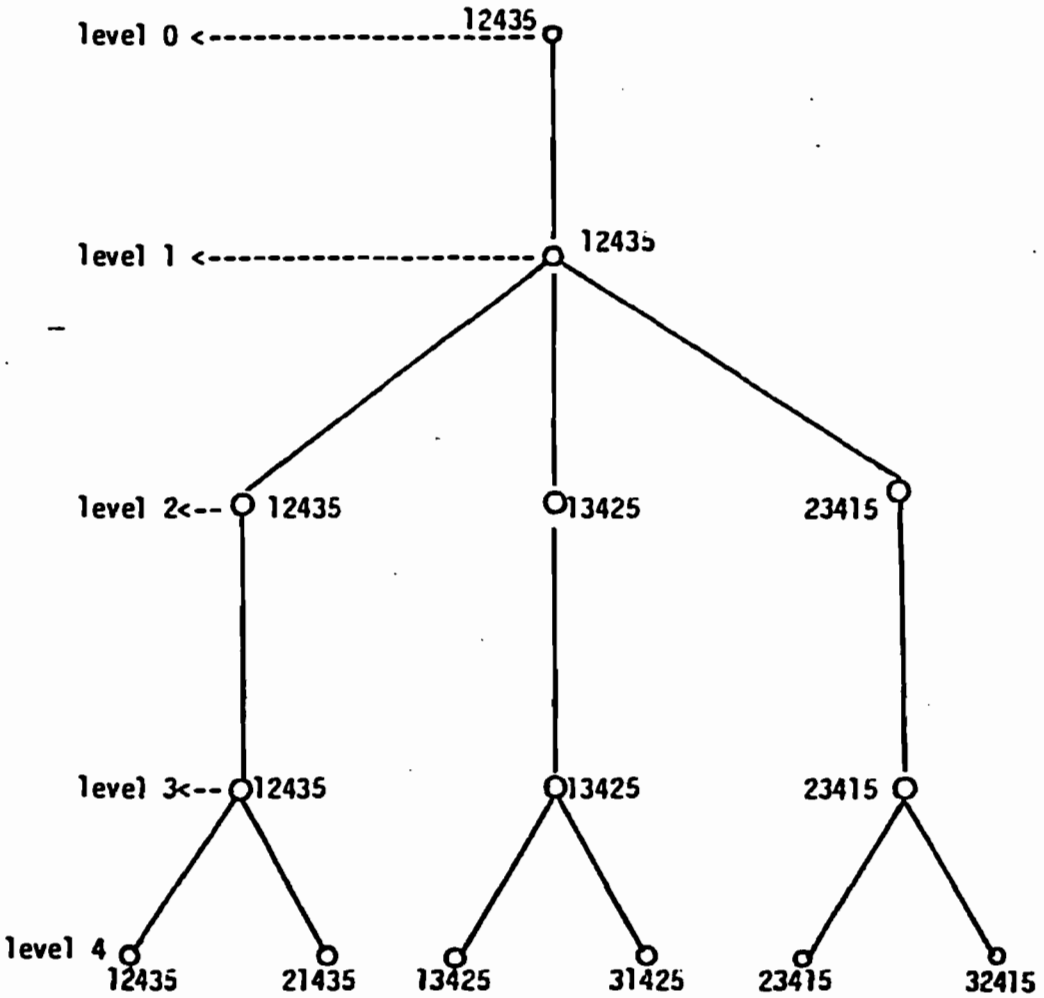
since the elements of $\hat{Z}_{m_i}$ are a subset of the subheap of x rooted at node (i+1),

$$y_j := \min(\hat{Z}_{m_i} - \{y_1,\ldots,y_{j-1}\}), \quad 2 \le j \le m_i .$$

In order to explain the above definition, let us consider the following example:

II.3.1 - Example:

x = 12435; hence, $T_5(x)$ is



$$\hat{Z}_{m_4} = \{5\}, \quad \hat{Z}_{m_3} = \{3\}, \quad \hat{Z}_{m_2} = \{4\}.$$

$$\hat{Z}_{m_1} = \{2,3\} \quad \text{and} \quad \hat{Z}_{m_0} = \{1,2,3\}.$$

From the above example we notice the following:

a) From level 1: x(5) = 5 must not label node 2 in x since the skewness is zero;

b) From level 3: in $T_5(x)$ the set of possible labels for node 2 in x is $\hat{Z}_{m_1}$ since the skewness is two;

c) From level 4: in $T_5(x)$ we notice that x(3) = 4 cannot label node 1 in x since in level 3 in $T_5(x)$ none of the heaps of size 2 contain 4, so $\hat{Z}_{m_0} = \{1,2,3\}$.

Now we define the corresponding sequences

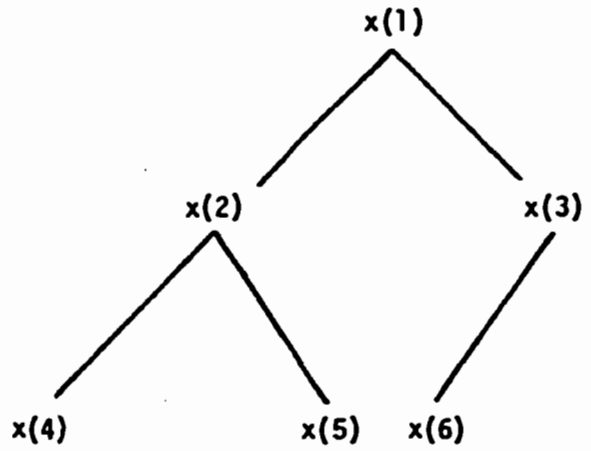
$$y_{m_4} := 5, \quad y_{m_3} := 3, \quad y_{m_2} := 4, \quad y_{m_1} := 23$$

and

$$y_{m_0} := 123.$$

As a more general case, consider the following examples:

II.3.2 - Example: Let N := 6



then the set of second order trees can be constructed in the following way:

since $x(6) < \infty$, $x(6)$ must label node 3, hence $\hat{Z}_{m_2} = \{x(3), x(6)\}$ and we know for any heap that $x(3) < x(6)$ implies $y_{m_2} = x(3)x(6)$.

Also, we have two possibilities of $\hat{Z}_{m_1}$, viz,

a) $x(5) > x(4)$.

or

b) $x(5) < x(4)$

in case $x(5) > x(4)$, we have $\hat{Z}'_{m_1} = \{x(2), x(4)\}$, $y'_{m_1} = x(2)x(4)$. In case

$$x(5) < x(4),$$

we have $\hat{Z}''_{m_1} = \{x(2), x(4), x(5)\}$ and $y''_{m_1} = x(2)x(5)x(4)$.

Finally, the set of all possible sequences which contain the set of all possible labels for node 1 in x is:

$F = \{$1. $x(1)x(2)x(4)$, 2. $x(1)x(3)x(6)x(5)x(4)$,
3. $x(1)x(2)x(3)x(6)x(4)$ 4. $x(1)x(3)x(2)x(6)x(4)$,
5. $x(1)x(3)x(2)x(4)$, 6. $x(1)x(2)x(3)x(4)$,
7. $x(1)x(3)x(6)x(2)x(5)x(4)$ 8. $x(1)x(2)x(3)x(6)x(5)x(4)$
9. $x(1)x(2)x(5)x(3)x(6)x(4)$ 10. $x(1)x(3)x(2)x(6)x(5)x(4)$
11. $x(1)x(3)x(2)x(5)x(6)x(4)$ 12. $x(1)x(2)x(3)x(5)x(6)x(4)$
13. $x(1)x(3)x(2)x(5)x(4)$ 14. $x(1)x(2)x(3)x(5)x(4)$
15. $x(1)x(2)x(5)x(3)x(4)$ 16. $x(1)x(2)x(5)x(4)\}$.

If we consider the above set F we notice the following:

a) The elements of y_{m_2} may not label node 1 in x, this can be seen from 1. x(1)x(2)x(4) in case x(3) > x(4);

b) Part of y_{m_2} may label node 1 in x, for example 15. x(1)x(2)x(5)x(3)x(4) in case x(6) > x(4);

c) Part of y_{m_2} may label node 1 in x, but may have different relation with the labels of node 2, for example 14. x(1)x(2)x(3)x(5)x(4) in case x(6) > x(4) and x(2) < x(3) < x(5). This means for any sequence y_{m_1} and y_{m_2} we have different sets of sequences which label node 1 in x. [Note $y_{m_1} = \{y'_{m_1}, y''_{m_1}\}$];

d) y_{m_2} may label node 1 in x for example 2. x(1)x(3)x(6)x(2)x(4).

The above example motivates the following:

II.3.3 - DEF: Let $h_{1:m}$, $y_{1:n}$ be any two sequences of positive integers with $h_1<\ldots<h_m$ and $y_1<\ldots<y_n$ where $h_{1:0}$ is the empty string. Define the operator $\overset{h_0}{\wedge}$ for $h_{1:m}$ and $y_{1:n}$ as follows $h_{1:m} \overset{h_0}{\wedge} y_{1:n} := \{h_0 z_{1:\ell_j} h_m$; $h_0 < y_1$ and $h_0 < h_1$ and $z_{1:\ell_j}$ is obtained from $h_{1:m}$ and $y_{1:j}$ by merging these sequences into ascending order, $0 \leq j \leq n\}$. Less formally the above set is the set of all possibilities to insert some part of $y_{1:n}$ in between $h_{1:m}$ with the property that every element is greater than the preceeding element.

II.3.4 - DEF: Let L_1, L_2 be any two sets of sequences of positive integers such that every element $h \in L_1$, $y \in L_2$ can be written as $h = h_{1:m}$ and $y = y_{1:f}$ with $h_1<\ldots<h_m$, $y_1<\ldots<y_f$ define $L_1 \overset{w}{\wedge} L_2 = \{t : t \in h \overset{w}{\wedge} y$ for some $h \in L_1$, $y \in L_2$ with $0 < w < \min\{h_1, y_1\}\}$.

This construction may be explained as follows:

Suppose w is the label for the root of the heap and $L_1(L_2)$ represents the set of all sequences labeling the root of the left (right) subheaps; then $L_1 \overset{w}{\wedge} L_2$ represents all possible labels for the root of the heap. Note that the sequences in $L_1(L_2)$ vary in length since some of the labels in a heap of size N may not label their fathers according to the construction of a second order tree. Note examples II.3.1 and II.3.2.

Now define the following sets recursively.

$$L_N^{(i)} := \{x(i);\ x \in H_{iN}\},\ \text{if } i \text{ is a leaf w.r.t. } N.$$

$$L_N^{\lfloor N/2 \rfloor} := \begin{cases} \{x(\lfloor N/2 \rfloor)x(N);\ x \in H_{iN}\}, & \text{if } N \text{ is even,} \\ \{x(\lfloor N/2 \rfloor)x(N-1),\ x(\lfloor N/2 \rfloor)x(N)x(N-1);\ x \in H_{iN}\}, & \text{if } N \text{ is odd,} \end{cases}$$

and for any node i define

$$L_N\lfloor N/2 \rfloor := L_N^{(2i)} \overset{x(i)}{\wedge} L_N^{(2i+1)} \quad \text{with } 2i+1 \leq N \text{ and } x \in H_{iN}.$$

$L_N\lfloor N/2 \rfloor$ is a set containing one sequence if N is even, or a set containing two sequences if N is odd.

II.3.5 - Theorem: - The number of the second order tree generated from H_N is

$$|J_N| = |L_N^{(1)}|.$$

Proof: By induction on N.

For $N = 2$ $|L_2^{(1)}| = 1$, $N = 3$, $|L_3^{(1)}| = 2$. Now assume the assertion is corre for all values of $K < N$ and let $x \in H_N$, $x_\ell(x_R)$ be the left (right) subheap of x, respectively. Assume size $(x_\ell) = M$, then size $(x_R) = N-M-1$.

By induction hypothesis $|J_M| = |L_M^{(2)}|$ and

$$|J_{N-M-1}| = |L_{N-M-1}^{(1)}| = |L_N^{(3)}|.$$

This implies for any h ϵ $L_N^{(2)}$ that the components of h represent the possible labels for node 2 generated from a subheap rooted at node 2, and that the length of h depend on how many elements from the subheap may label node 2 (note example II.3.2 and II.3.1). Moreover, for y ϵ $L_N^{(3)}$, h and y represent second order tree generated from the left or right subheap, respectively. But the element of y and h are a subsequence of {2,...,N} which means y and h may have different values but still represent the same second order tree generated from the left or right subheap. [This implies tha we can get different sets of possibilities for the relation between h and y. Moreover, we know from the construction of the second order tree that for every x(ℓ) ϵ h and x(r) ϵ y there exists at least one inverse image on level (N-3) of the second order tree. This inverse image contains exactly one heap of size 3; x(r) labels node 3, x(ℓ) labels node 2, and this is correct for all x(r) ϵ y, x(ℓ) ϵ h. If for a given x(r) ϵ y and for every x(ℓ) ϵ h, x(r) > x(ℓ) then the skewness for every inverse image contains x(r) as a label for node 3 is zero. x(r) in turn will not label node 1. But, if for some x(ℓ) ϵ h, x(r) < x(ℓ) then x(r) will label node 1, since the skewness for this inverse image at level (N-3) is 1. So for every x(r) ϵ y we have to compare it with every x(ℓ) ϵ h which gives us h $\overset{x(1)}{\wedge}$ y possibilities. But for every subheap x_ℓ we have ν(N-M-1) possibilities for x_R and this number ν(N-M-1) generates $|L_{N-M-1}^{(1)}| = |L_N^{(3)}|$ second order trees by induction. This means every y ϵ $L_N^{(3)}$ must be compared with every h ϵ $L_N^{(2)}$. This implies

$$|L_N^{(1)}| = |L_N^{(2)} \overset{x(1)}{\wedge} L_N^{(3)}| = |J_N| \cdot \square$$

One can see that the length of the sequences in $L_N^{(1)}$ is between $\lfloor \log_2 N \rfloor + 1$ and N. For example if the labels of the leftmost path for some X ϵ H_N are from an ascending chain 1,...,$\lfloor \log_2 N \rfloor + 1$, while we can get the maximum length N if for some X ϵ H_N every label in X has chance to label

II.3.6- Lemma: Let $C(N,\ell)$ be the number of the sequences of length ℓ in $L_N^{(1)}$; assume the left subheap has N elements, then we have the recursive relation

$$C(N,\ell) = \sum_{j\le \ell-1} \sum_{i=\ell-j-1} C(N,j)C(N-N-1,i)\ \binom{\ell-2}{j-1}$$

with

$$C(N,j) = 0 \quad \text{if} \quad j > N \ \text{ or } \ j \le \lfloor \log_2 N \rfloor .$$

Proof: Since $C(N,\ell)$ represents the number of the sequence of length ℓ in $L_N^{(1)}$ and since we know that x(1) is the label of the root and that this ele is the leading element of every sequence in $L_N^{(1)}$, we also know that every label of node 2 must label node 1 and some of the labels of node 3 may label node 1. We also know that from both sides we have to collect the sequences length i and j such that $i + j \ge \ell - 1$ and ($i \le \ell - 1$ holds since every element from the label of the left node (2) must label node 1), for each i we have to insert $(\ell-j-1)$ elements from the sequence of length j in between the element of i to get $\ell-1$ elements. We now have the label of the root x(1). This gives us the sequence of length ℓ. Now assume $h \in L_N^{(2)}$ and $|h| = i$; i.e., $h = h_{1:i}$. Hence we have to insert $\ell-j-1$ elements from the sequence of length j, say $y = y_{1:j}$, such that $y_{\ell-j-1} < h_i$. This insertion can be done in $\binom{\ell-2}{\ell-j-1}$. □

Now we are ready for a proof of Lemma II.3.3 on page 33.

Proof of Lemma II.3.3 on page 33:

By induction on N, N=3 hence: $|J_3^{max}| = 1$, $N = 4$, $|J_4^{max}| = 2 = |L_3^{(1)}|$. Assume the statement is correct for all values of N' smaller than N. Without loss of generality it is sufficient to show $|J_N^{max}| = |L_{N-1}^{(1)}|$ since $|J_{N-1}| = |L_{N-1}^{(1)}|$. Assume that the

insertion is done in the left subheap (i.e., N belongs to the left subtree of x) and assume that the size of x_2 is M. This implies

$$|J_M^{max}| = |L_{M-1}^{(1)}| = |L_{M-1}^{(2)}| .$$

and

$$|J_{M-M-1}^{max}| = |L_{M-1}^{(3)}| .$$

But x represents a heap the second order tree of which has a maximal number of offsprings. This implies $x(2) < x(3)$ which further implies that for any $h \in L_{M-1}^{(3)}$ and $y \in L_{M-1}^{(3)}$ the leading element in h is x(2) and the leading element in y is x(3). But $x(2) < x(3)$. This implies that x(2) is smaller than any element in y; therefore the order relation between x(2) and the element of y is fixed. The conclusion is that if the length of h is $\underline{m}$ we have to consider only m-1 elements of h and this is true for any $h \in L_{M-1}^{(2)}$. This implies the number of the second order tree with the maximum offspring is

$$|L_{M-1}^{(1)} \overset{x(1)}{\wedge} L_{M-M-2}^{(3)}| = |L_{M-1}^{(1)}| . \quad \square$$

II.4 Generating functions

II.4.1 In order to study the solution of II.3.6, we will study the case $N = 2^k - 1$ for some integer k.

Relation II.3.6 can be transformed into

$$C(k,\ell) = \sum_{j=k-1}^{\ell-1} \sum_{i=\ell-j-1}^{2^{k-1}-1} C(k-1,j)C(k-1,i)\binom{\ell-2}{j-1}. \qquad \text{II.4.1}$$

Define $C(k,\ell)$ as zero if $k > \ell$ or $\ell > 2^k - 1$, and $\binom{\ell-2}{\ell-j-1}$ is zero if $\ell-j-1 > \ell-2$.

Now let

$$C_k(t) := \sum_{\ell=0}^{\infty} \frac{C(k,\ell)t^\ell}{\ell!}$$

be the exponential generating function of $\{C(k,\ell)\}_{\ell \geq 0}$. This implies

$$\sum_{\ell=0}^{\infty} \frac{C(k,\ell)t^\ell}{\ell!} = \sum_{\ell=0}^{\infty} \sum_{j=k-1}^{\ell-1} \sum_{i=\ell-j-1} \frac{C(k,j)C(k,i)\binom{\ell-2}{j-1}t^\ell}{\ell!}. \qquad \text{II.4.1(a)}$$

Taking the second derivatives of both sides of II.4.1(a) we get

$$\sum_{\ell=0}^{\infty} \frac{C(k,\ell)\ell(\ell-1)t^{\ell-2}}{\ell!} = \sum_{\ell=0}^{\infty} \sum_{j=k-1}^{\ell-1} \sum_{i=\ell-j-1} \frac{C(k-1,i)C(k-1,j)t^{\ell-2}}{(j-1)!(\ell-j-1)!}$$

$$C_k''(t) = \sum_{\ell=0}^{\infty} \sum_{j=k-1}^{\ell} \sum_{i=\ell-j-1}^{2^{k-1}-1} \frac{C(k-1,i)C(k-1,j)t^{\ell-2}}{(j-1)!(\ell-j-1)!}$$

$$= \sum_{\ell=0}^{\infty} \sum_{j=k-1}^{\infty} \sum_{i=\ell-1}^{2^{k-1}-1} \frac{C(k-1,j)C(k-1,i)t^{\ell+j-2}}{(j-1)!(\ell-1)!} \qquad (**)$$

$$= \sum_{j=k-1}^{\infty} \frac{C(k-1,j)t^{j-1}}{(j-1)!} \cdot \sum_{\ell=0}^{\infty} \sum_{i=\ell-1}^{2^{k-1}-1} \frac{C(k-1,i)t^{\ell-1}}{(\ell-1)!}$$

$$= C'_{k-1}(t) \cdot \sum_{\ell=0}^{\infty} \sum_{i=\ell-1}^{2^{k-1}-1} \frac{C(k-1,i)t^{\ell-1}}{(\ell-1)!}$$

$$= C'_{k-1}(t) \cdot \sum_{i=0}^{2^{k-1}-1} C_{k-1}^{(i)}(t) \qquad \text{II.4.1(b)}$$

(here $C_k^{(i)}(t)$ is, as usual, the ith derivatives of $C_k(t)$). Let

$$g_k = \sum_{i=0}^{2^k-1} C_k^{(i)}(t);$$

this implies relation II.4.1(b) can be transformed into the following relation;

$$g''_k - g'^{''}_k = (g'_{k-1} - g''_{k-1})g_{k-1} \cdots \qquad \text{II.4.1(c)}$$

(**) ([McB], relation (13), p. 6).

—

II.4.2 - Corollary:

The number $|J_N|$ of second order trees generated from the set of heaps of size N is

$$\sum_j \sum_i C(k-1,j)C(k-1,i)\binom{i+j}{j},$$

where $N = 2^k - 1$.

Proof: Assume C(k-1,j) represents the number of sequences of length j in the set of possible labels for node 2 and C(k-1,i) represents the same thing for node 3. The cardinality of the set of all sequences we may obtain from these two sequences is

$$C(k-1,j)C(k-1,i)\binom{i+j}{j},$$

and these sequences vary in length. Take the sum over all j and over all i to obtain the cardinality of $L_N^{(1)}$, which is the number of second order trees.

II.5 Solution for some terms of the nonlinear differential difference equation

Consider the case $\ell=k$ in II.4.1, which implies that

$$C(k,k) = C(k-1,k-1)|J_{2^{k-1}-1}|$$

which in turn suggests

$$C(k,k) = |J_{2^{k-1}-1}| \cdot |J_{2^{k-2}-1}| \cdots |J_3| \cdot |J_1|. \qquad \text{II.5.1}$$

Also, if $\ell = k + 1$ in II.4.1 we get

$$C(k,k+1) = \frac{k(k-1)}{2} C(k,k). \qquad \text{II.5.2}$$

If $\ell = 2^k-1$ in II.4.1 we get

$$C(k,2^k-1) = C(k-1,2^{k-1}-1)\binom{2^k-3}{2^{k-1}-2}$$

$$= \frac{(2^k-3)!}{\prod_{i=0}^{k-3} (2^{k-i-1}-1)^{2^i} (2^{k-i-1}-2)^{2^{i+1}}} \qquad \text{II.5.3}$$

From II.5.3 and II.4.1 we get

$$C(k,2^k-2) = \frac{h(k)\ (2^k-4)!}{\prod_{i=0}^{k-3} (2^{k-i-1}-1)^{2^i} (2^{k-i-1}-2)^{2^{i+1}} (2^{i+2}-3)}, \qquad \text{II.5.4(a)}$$

where

$$h(k) = h(k-1)(2^k-3) + (2^{k-1}-1)(2^{k-1}-2)(2^{k-2}-2)\ldots \quad 13\text{-}5\text{-}1$$

This implies

$$h(k) = \sum_{j=2}^{k} \frac{(2^{k-j+1}-1) \prod_{i=0}^{k-2} (2^{k-i}-3)}{(2^{k-j+2}-3)}. \qquad \text{II.5.4(b)}$$

From II.5.4(a) and II.5.4(b) we get

$$C(k,2^k-2) = C(k,2^k-1) \cdot \sum_{j=2}^{k} \frac{2^{k-j+1}-1}{2^{k-j+2}-3} \qquad \text{II.5.5}$$

$$C(2^k-3) = \sum_{i=2}^{k-1} f'(i+1)b(i)c(i) \prod_{j=i}^{k-1} c(j) \prod_{j=i+1}^{k} f''(j+1),$$

where

$$f'(i) := \binom{2^i-5}{2^{i-1}-2}$$

$$b(k) := \left(\sum_{j=2}^{k} \frac{2^{k-j+1}-1}{2^{k-j+2}-3} + 1 \right)^2$$

$$c(k) := \frac{(2^k-3)!}{\prod_{i=0}^{k-3} (2^{k-i-1}-1)^{2^i} (2^{k-i-1}-2)^{2^{i+1}}}$$

and

$$f''(j) := \begin{cases} 1 & \text{if } j = k+1 \\ \binom{2^j-4}{2^{j-1}-2} & \text{if } j \leq k \end{cases} .$$

II.6 Asymptotic expansion for a quantity related to the nonlinear differential equation

Rewrite II.5.3 in the following way:

$$\ln C(k,2^k-1) = \ln((N-2)!) - \sum_{i=0}^{k-3}[2^i \ln(2^{k-i-1}-1) + 2^{i+1}\ln(2^{k-i-1}-2)] = \ln((N-2)!) - \sum_{i=0}^{k-3}[\underbrace{2^{k-3-i}\ln(2^{i+2}-1)}_{(a)} + \underbrace{2^{k-2-i}\ln(2^{i+2}-2)}_{(b)}].$$

$N = 2^k-1$ it implies

$$(a)\quad \sum_{i=0}^{k-3} 2^{k-3-i}\ln(2^{i+2}-1) =$$

$$= \sum_{i=0}^{k-3} 2^{k-3-i}[(i+2)\ln 2 + \ln(1 - 2^{-(i+2)})]$$

$$= \sum_{i=0}^{k-3} 2^{k-3-i}(i+2)\ln 2 + \sum_{i=0}^{k-3} 2^{k-3-i}\ln(1-2^{-(i+2)})$$

$$= \ln 2[2^k-2k] + \sum_{i=0}^{k-3} 2^{k-3-i}\ln(1-2^{-(i+2)}); \qquad \text{II.6.1}$$

but,

$$\sum_{i=0}^{k-3} 2^{k-3-i}\ln(1-2^{-(i+2)}) = 2^{k-3}\sum_{j=1}^{\infty}\sum_{i=0}^{k-3}\frac{1}{j2^{2j}2^{i\cdot j}2^i}$$

-2^{k-3} [Part I - Part II], where

$$\text{Part I} = -2^{k-3}\sum_{j=1}^{\infty}\frac{1-2^{-(k-2)(j+1)}}{j2^{2j}(1-2^{-(j+1)})} = -2^{k-3}\sum_{j=1}^{\infty}\frac{1}{j2^{2j}(1-2^{-(j+1)})};$$

$$\text{Part II} = \sum_{j=1}^{\infty}\frac{2^{-(k-2)(j+1)}}{j2^{2j}(1-2^{-(j+1)})}.$$

We can also show that part I is some constant by taking partial fractions:

$$\sum_{j=1}^{\infty} \frac{1}{j2^{2j}(1-2^{-(j+1)})} = 4 \sum_{j=1}^{\infty} \frac{1}{j} \{\frac{1}{2^{j+1}-1} - \frac{1}{2^{j+1}}\}$$

$$= 4 \sum_{t=2}^{\infty} \frac{\ln(1-2^{-t})}{2^t} . \qquad \text{II.6.2(a)}$$

Part (II): $\sum_{j=1}^{\infty} \frac{2^{-(k-2)(j+1)}}{j2^{2j}(1-2^{-(j+1)})} = 4 \sum_{j=1}^{\infty} \frac{2^{-(k+1)(j+1)}}{j(2^{j+1}-1)} .$

Let

$$a_k = \sum_{j=1}^{\infty} \frac{2^{-(k+1)(j+1)}}{j(2^{j+1}-1)} .$$

Then the generating function for $\{a_k\}$ satisfies

$$\sum_{k=0}^{\infty} a_k z^k = \sum_{j=1}^{\infty} \frac{1}{j(2^{j+1}-1)(2^{j+1}-z)} ;$$

by Darboux's theorem,

$$a_k = \frac{1}{3\ 4^{k+1}} + O(4^{-k}). \qquad \text{II.6.2(b)}$$

This implies

$$\sum_{i=0}^{k-3} 2^{k-3-i} \ln(2^{i+2}-1) = \ln 2[2^k-2k] -$$

$$2^{k-3}[4 \sum_{t=2}^{\infty} \frac{\ln(1-2^{-t})}{2^t} - 4(\frac{1}{3\ 4^{k+1}} + o(4^{-k})]$$

$$= \ln 2(2^k-2k) + 4\alpha_2\ 2^{k-3} + \frac{2^{-k-3}}{3} + O(2^{-k}),$$

where

$$\alpha_2 = -\sum_{t=2}^{\infty} \frac{\ln(1-2^{-t})}{2^t} . \qquad \text{II.6.3}$$

Part (b) $$\sum_{i=0}^{\infty} 2^{k-2-i}\ln(2^{i+2}-2) = \sum_{i=0}^{k-3} 2^{k-2-i}[(i+2)\ln 2 + \ln(1-2^{-(i+1)})]$$

$$= (2^{k+1}-4k)\ln 2 + \alpha_1 2^{k-2} + \frac{2^{-k+5}}{3} + O(2^{-k}),$$

where

$$\alpha_1 = \sum_{t=1}^{\infty} \frac{\ln(1-2^{-t})}{k^{2t}} .$$

This implies

$$(a)+(b) = \ln 2(3\cdot 2^k - 6k) + (\alpha_1+2\alpha_2)2^{k-2} + 83.2^{-k-3} + o(2^{-k}). \quad \text{II.6.4}$$

So that, in turn,

$$C(k,2^k-1) = (2\pi(2^k-1))^{1/2}\left(\frac{2^k-1}{e}\right)^{2^k-1} \cdot \exp[\ln 2(3\cdot 2^k-6k)$$

$$+ (\alpha_1+2\alpha_2)2^{k-2} + 83.2^{-k-3}] + O\left(\frac{1}{2^k-1}\right). \qquad \square$$

II.7 Numerical Computation Using Lemma II.3.6 we obtain Tables 1, 2, and 3.

size 7	C's	size 15	C's
C(3,3)	2	C(4,4)	92
C(3,4)	6	C(4,5)	552
C(3,5)	12	C(4,6)	2208
C(3,6)	16	C(4,7)	7176
C(3,7)	10	C(4,8)	19996
		C(4,9)	48832
		C(4,10)	105280
		C(4,11)	198624
		C(4,12)	318336
		C(4,13)	407352
		C(4,14)	366960
		C(4,15)	171600
TOTAL J_7	46	J_{15}	1647008

Table 1: The number of the second order trees generated from H_7 and H_{15}.

size 31	
C(5,5)	151524736
C(5,6)	1515247360
C(5,7)	4697583104
C(5,8)	50306212352
C(5,9)	229158105088
C(5,10)	950599839360
C(5,11)	3661116119552
C(5,12)	13239678820928
C(5,13)	45261947438576
C(5,14)	146865564155488
C(5,15)	453334051796032
C(5,16)	1332821328117504
C(5,17)	3735397350436080
C(5,18)	9966795061001160
C(5,19)	25486772330436912
C(5,20)	62122817930866816
C(5,21)	144603599623470080
C(5,22)	320990758912925888
C(5,23)	677052654083328000
C(5,24)	1347468403865374720
C(5,25)	2500351977366586544
C(5,26)	4245409076231541504
C(5,27)	6410880385776979200
C(5,28)	8246153711771712000
C(5,29)	9526463181334272000
C(5,30)	9065197045839360000
C(5,31)	7283838679865600000
TOTAL J_{31}	38371737481952002304

Table 2: The number of the second order trees generated from H_{31} .

N	J_N
1	1
2	1
3	2
4	3
5	7
6	16
7	46
8	130
9	411
10	1314
11	4636
12	20790
13	84962
14	376826
15	1647008
31	38871737481952002304

Table 3: N represent the heap size and J_N is the number of the second order trees.

CHAPTER III

UPPER BOUND FOR THE NUMBER OF THE SECOND ORDER TREE

III.0 Introduction We have seen in Chapter II that the exponential generating function of the number of second order trees satisfy a non-linear differential difference equation. In this chapter we will prove that the number of the second order tree is bounded above by some function related to the number of heaps and the height of heap. This upper bound is given in the following theorem and its corollary.

III.1 Computation of the upper bound

III.1.1-Theorem: Let $x \in H_N$ and assume T_N is the second order tree generated from x; define m as

$m := \{i;$ i satisfies the following conditions:

a) $i < N$, b) i is even and $2i > ii$, c) $i+1 \leq N$, and
d) $x(i) < x(i+1)\}$.

Let $x_m := \{x(i);\ i \in m\}$, define Y as:

$$Y := (y_1, y_2, \ldots, y_{|m|}),$$

where

$$y_1 := \min x_m$$
$$y_2 := \min(x_m - \{y_1\})$$
$$\vdots$$
$$y_j := \min(x_m - \{y_1, y_2, \ldots, y_{j-1}\})$$

for all j, $2 \leq j \leq |m|$.

Then, the multiplicity of T_N (i.e., $\mu(T_N)$) is bounded from below as follows

$$\mu(T_N) \geq \begin{cases} 1 & \text{if } m = \phi \\ \max\left\{\frac{(N-y_1)!}{(N-y_1-1)!}, \frac{(N-y_2)!}{(N-y_2-2)!}, \ldots, \frac{(N-y_{|m|})!}{(N-y_{|m|}-|m|)!}\right\}, & \text{if } m \neq \phi \end{cases}.$$

Proof:

$m = \phi$ is trivial since x generates T_N.

$m \neq \phi$ For simplicity, assume for the moment m={i}; hence, x(i)<x(i+1). This implies that if we look at level N-i-1 in T_N, then the root of every subtree rooted at that level is unary, which in turn means T_N is determined by an order relation ρ on the components of x and that one of the relations is "x(i) < x(i+1)". Restrict ρ to the components of x without x(i+1) and call this restriction ρ'. The reason for this restriction follows:

Assume T_N has the set of all permutations of size N which generate x as labels for its nodes, and suppose the labels on level j are

$$x_{j,1},\ldots,x_{j,k_j} \qquad (0 \leq j \leq N-1)$$

[thus, $x_{j,\ell}$ is a permutation of {1,...,N} such that $x_{j,\ell}(1),\ldots,x_{j,\ell}(N-j)$ form a heap and after j iterations, $x_{j,\ell}$ will become x under Williams' algorithm for $1 \leq \ell \leq k_j$]. Now we note the following:

a) i+1 is odd and a leaf; this implies

$$i+1 \notin \{t(N-j,m),\ 0 \leq m \leq \lfloor \log(N-j) \rfloor\} \cup \{b(N-j,m),\ 0 \leq m \leq \lfloor \log(N-j) \rfloor$$

for any $j < N - i - 1$,

which means x(i+1) does not affect the calculation of the skewness of any $x_{j,\ell}$ $0 \leq j \leq N - i - 1$ and $1 \leq \ell \leq k_j$.

b) Since x(i+1) > x(i), x(i+1) must not label its parent, which means x(i+1) is not one of the components of

$$x_{j,\ell}(1),\ldots,x_{j,\ell}(N-j) \quad N-i \leq j \leq N-1, \quad 1 \leq \ell \leq k_j.$$

But, x is a heap of size N; therefore, x(i+1) has three possibilities:

1. x(i+1) = x(i) + 1;
2. x(i+1) = N;

or

3. x(i)+2 ≤ x(i+1) < N;

1. If x(i+1) = x(i)+1. Define $x^{(1)}$ as x and and let $x^{(2)}$ be the permutation generated from interchanges of x(i+1) and x(i+1) + 1 in $x^{(1)}$. In other words, define $x^{(2)}$ as

$$x^{(2)}(k) := \begin{cases} x^{(1)}(\ell), & k = i+1 \\ x^{(1)}(i+1), & k=\ell \\ x^{(1)}(k), & \text{otherwise} \end{cases}$$

Claim:

α) $x^{(2)}$ is a heap.

β) $x^{(2)}$ and $x^{(1)}$ generate the same second order tree.

Proof of this claim:

α) It is clear that $x^{(2)}(i+1) > x^{(2)}(\lfloor \frac{i+1}{2} \rfloor)$ since $x^{(2)}(i+1) = x^{(1)}(\ell) = x^{(1)}(i+1) + 1$, also $x^{(2)}(\ell) > x^{(2)}(\lfloor \ell/2 \rfloor)$ and because we know from the construction of $x^{(2)}$ that the difference between $x^{(1)}(i+1)$ and $x^{(1)}(\ell)$ is 1. Now, every component in the subheap of $x^{(2)}$ rooted at ℓ has a greater label than $x^{(2)}(\ell)$.

c) Since the order is the only important relation in constructing second order trees, we have

$$x^{(2)}_{(i+1)} > x^{(2)}(i).$$

We also have originally ρ' = order relation on $\{x - \{x(i+1)\}\}$ = order relation on $\{x^{(1)} - \{x^{(1)}(i+1)\}\}$. This implies that if we decrease each component in $\{x^{(1)} - \{x(i+1)\}\}$ which is greater than $x^{(1)}(i+1)$ by one and decrease every component in $\{x^{(2)} - \{x^{(2)}(i+1)\}\}$ which is greater than $x^{(2)}(i+1)$ by 1, we get the same sequence. This can be seen in the following way:

Define

$Y^{(1)}$ from $x^{(1)}$ as

$$Y^{(1)} = x^{(1)}(1)x^{(1)}(2) \ldots x^{(1)}(i)x^{(1)}(i+2) \ldots x^{(1)}(\ell) \ldots x^{(1)}(l)$$

i.e. $Y^{(1)}$ is exactly $x^{(1)}$ after removing the (i+1)th component.

Define $Y^{(2)}$ from $x^{(2)}$ as

$$Y^{(2)} = x^{(2)}(1) \ldots x^{(2)}(i)x^{(2)}(i+2) \ldots x^{(2)}(\ell) \ldots x^{(2)}(l).$$

After decreasing every element in $Y^{(1)}$ greater than $x^{(1)}(i+1)$, assume we get $Z^{(1)}$. Then we have

$$Z^{(1)}(k) = \begin{cases} x^{(1)}(k)-1 & \text{if } x^{(1)}(k)>x^{(1)}(i+1) \text{ and } x^{(1)}(k) \text{ is in } Y^{(1)} \\ x^{(1)}(k) & \text{if } x^{(1)}(k)<x^{(1)}(i+1) \text{ and } x^{(1)}(k) \text{ is in } Y^{(1)} \end{cases}$$

and the same for $x^{(2)}$; therefore, assume we get $Z^{(2)}$. This implies

$$Z^{(2)}(k) = \begin{cases} x^{(2)}(k)-1 & \text{if } x^{(2)}(k)>x^{(2)}(i+1) \text{ and } x^{(2)}(k) \text{ is in } Y^{(2)} \\ x^{(2)}(k) & \text{if } x^{(2)}(k)<x^{(2)}(i+1) \text{ and } x^{(2)}(k) \text{ is in } Y^{(2)}. \end{cases}$$

We have to show now that

$$Z^{(2)} = Z^{(1)}$$

for all k such that

$$k \in \{1,\ldots,n\} - \{i+1\}$$

For every $Z^{(2)}(k)$ we have the following cases:

Case I $x^{(2)}(k) < x^{(2)}(i+1)$ and $x^{(2)}(k) \neq x^{(2)}(\ell)$, which implies $x^{(2)}(k) < x^{(1)}(i+1)$; this in turn implies $Z^{(2)}(k) = Z^{(1)}(k)$.

Case II $x^{(2)}(k) < x^{(2)}(i+1)$ and $x^{(2)}(k) = x^{(2)}(\ell)$.

But $x^{(1)}(\ell) = x^{(1)}(i+1)$; this implies $Z^{(1)}(\ell) = x^{(1)}(i+1) =$ $= x^{(2)}(\ell) = Z^{(2)}(k)$.

Case III $x^{(2)}(k) > x^{(2)}(i+1)$, which implies $x^{(2)}(k) > x^{(1)}(i+1)$; hence, this implies $Z^{(2)}(k) = Z^{(1)}(k)$, which in turn implies

$$Z^{(2)} = Z^{(1)}.$$

All this implies that ρ' is satisfied over $x^{(2)}-\{x^{(2)}(i+1)\}$. Consequently, ρ remains invariant in $x^{(2)}$.

Now, from $x^{(2)}$ construct $x^{(3)}$ by interchanging $x^{(2)}(i+1)$ with $x^{(2)}(i+1)+1$ and repeat the same process until $x^{(n)}(i+1) = N$, where $n = N - x(i)$.

2.) If $x(i+1) = N$, define $x^{(1)} = x$ and $x^{(2)}$ from $x^{(1)}$ as

$$x^{(2)}(i+1) = x^{(1)}(i+1)-1 = x^{(1)}(\ell); \quad x^{(2)}(\ell) = x^{(1)}(i+1)$$

and $x^{(2)}(k) = x^{(1)}(k)$ for $k \notin \{i+1,\ell\}$. Use the same proof in 1.) and repeat the same process until $x^{(n)}(i+1) = x^{(1)}(i) + 1$.

3.) can be partitioned into 1.) and 2.).

If $|m| > 1$, we can use the same techniques; but, if $|m| = j$, then we have $N-y_j$ elements greater than y_j. This implies we have $\binom{N-y_j}{j}$ possible choices for the brothers of the elements of m and for each choice we have j! arrangements. In other words, we have at least $\frac{(N-y_j)!}{(N-y_j-j)!}$ possibilities of heaps generating the same second order trees. □

III.1.2 - Corollary: Let $N = 2^k-1$.

$$f(k) = 2^{k-2} \cdot 2^{2^{k-2}-1}\left[(2^k-k-2)\binom{2^{k-1}-1}{k-1} - (k-1)\binom{2^{k-1}-1}{k}\right]$$

$$+ |H_{2^{k-1}-1}|\,2^{k-2}\left[5\cdot 3^{2^{k-2}-2}(2^{k-2}-1) - 2^{2^{k-2}-1}(2^{k-1}-2)\right]$$

$$+ |H_{2^{k-1}-1}|\,2^{k-2}\left[(2^{k-2}-2)^2 - \frac{(2^{k-2}-2)(2^{k-2}-3)}{2}\right].$$

Then the number $|J_{2^k-1}|$ of second order trees generated from H_{2^k-1} (the set of heaps of size 2^k-1) is bounded from above by $|H_{2^k-1}| - f(k)$; thus,

$$|J_{2^k-1}| \leq |H_{2^k-1}| - f(k). \qquad (*)$$

Proof: The proof for the above relation (*) is a technical computation depending on theorem III.1.1 and the set of all possible labels for the leaves in any heap of size 2^k-1. For simplicity we will break up the proof into some simpler steps.

1) In theorem III.1.1 let $|m| = 1$; hence, the calculation of the multiplicity of a given second order tree is done according to the two leaves having the same parent.

2) It follows from the definition of heaps that the set of all possible labels of the element of m must be taken from the set A, where A is

3) Partition A into the following three subsets:

a) $A_1 = \{k,\ldots,2^{k-1}\}$;

b) $A_2 = \{2^{k-1}+1,\ldots,3\cdot 2^{k-2}-1\}$;

c) $A_3 = \{3\cdot 2^{k-2},\ldots,2^k-1\}$.

a): Computation according to A_1.

a.1) For every $j \in A_1$, define the set $A_1^{(j)}$ as

$$A_1^{(j)} := \{x^{(j)};\ x^{(j)} \text{ is a heap of size } 2^k-1$$

and

$x^{(j)}$ satisfy the following properties:

(1) j is the label of the element in m;

(2) the label of the right offspring of the element of m is greater than j;

(3) each element of $\{1,\ldots,j-1\}$ labels some internal node of $x^{(j)}$.$\}$

Claim 1: $A_1^{(j)}$ contains at least

$$\binom{j-2}{k-2}2^{k-2}\ 2^{2^{k-2}-1}(N-j) \text{ heaps.}$$

Proof: From theorem III.1.1 we know that every second order tree generated from $x^{(j)} \in A_1^{(j)}$ has a multiplicity of at least N-j. For every one of these heaps, the path leading from the root to the node with label j contains k nodes and the label for these nodes must be taken from the set $\{1,\ldots,j\}$, 1 is the root label and j is the label for the node in m. Thus we have at least $\binom{j-2}{k-2}$ possible choices for labeling this path.

For every choice we can rearrange the rest of the elements to satisfy the conditions (1), (2), and (3) because we have $2^{k-1}-1$ internal nodes. Also, for any one of the above heaps we have 2^{k-1} leaves, for any two leaves having the same parent (except the node in m and its brother). If we interchange the label of the right offspring with the labels of the left offspring, we get a new heap satisfying conditions (1), (2), and (3); since we have $2^{k-2}-1$ pairs of leaves, we have the following number of heaps

$$\sum_{i=0}^{2^{k-2}-1} \binom{2^{k-2}-1}{i} = 2^{2^{k-2}-1} .$$

The index i in this summation stands for the number of pairs we have to interchange. Finally, since m can take any node of the form $2^{k-1} + 2i$, $0 \leq i \leq 2^{k-2}-1$, each time we label the node in m by j and condition (1), (2) and (3) is satisfied, we will get $\binom{j-2}{k-2} 2^{2^{k-2}-1}\ 2^{k-2}(n-j)$ heaps.

Claim 2: If $j_1, j_2 \in A_1$, then

$$A_1^{(j_1)} \cap A_1^{(j_2)} = \phi \text{ if } j_1 \neq j_2 .$$

Proof: It is immediate from condition (3) since if $j_1 < j_2$, then j_1 is a leaf in any $x^{(j_1)} \in A_1^{(j_1)}$ and j_1 is a label for some internal node in $x^{(j_2)} \in A_1^{(j_2)}$.

Claim 3: There exists

$$\sum_{j \in A_1} \binom{j-2}{k-2} 2^{2^{k-2}-1}\ 2^{k-2}\ (n-j)$$

heaps generating at most

$$\sum_{j \in A_1} \binom{j-2}{k-2} 2^{2^{k-2}-1}\ 2^{k-2} \text{ second order trees.}$$

Proof: Since N-j is the multiplicity of any second order tree generated from any $x \in A_1$, the result follows.

b): Computation according to A_2.

Define for every $j \in A_2$, $A_2^{(j)}$ as $A_2^{(j)} = \{x^{(j)}, x^{(j)}$ is a heap of size 2^k-1 and $x^{(j)}$ satisfies the following conditions:

(1) the label of the node in n is j and the label of the brother of this node is greater than j;

This definition is rather long, on the other hand right and left leaves are somewhat obvious. Couldn't you

* make the box a footnote
* omit it in the text of the def?

(2) the label of the internal nodes is an element of $\{1,\ldots,2^{k-1}-1,\}$;

(3) each element of $\{2^{k-1},\ldots,j-1\}$ labels some right leaves+ in $x^{(j)}$. Then, the labels for the left leaves must be chosen from the set $\{j+1,\ldots,2^k-1\}$.

Claim 4: The cardinality of $A_2^{(j)}$ is at least

$$|H_{2^{k-1}-1}|\ 2^{k-2}\ 2^{32^{k-2}-j-1}\binom{2^{k-2}-1}{j-2^{k-1}}[N-j] \text{ for every } j \in A_2.$$

Proof: For every second order tree generated from $x \in A_2^{(j)}$ there exists at least N-j heaps generating this tree. This follows from theorem III.1.1. For every one of these heaps the labels of the internal nodes are smaller than the labels of the leaves. Since the number of the internal nodes is $2^{k-1}-1$ and the number of heaps of size $2^{k-1}-1$ is $|H_{2^{k-1}-1}|$, we can inter-

+Define the leaf [from box].

change the labels of the internal nodes to get at least $(k-j)!\,!_{2^{k-1}-1}$ heaps satisfying condition (1), (2), and (3). Also, due to condition (3) for every heap from above we have 2^{k-2} right leaves. One of them is the right offspring for the node in m, so we have to label $j-2^{k-1}$ of them by the set $\{2^{k-1},\ldots,j-1\}$. Thus, we have $\binom{2^{k-2}-1}{j-2^{k-1}}$ possibilities for every heap from above. If we interchange between the labels of the leaves having the same parent, except for those leaves which satisfy conditions (1) and (3), we will get

$$2^{k-2}-\sum_{i=0}^{(j-2^{k-1}+1)}\binom{2^{k-2}-(j-2^{k-1}+1)}{i} = 2^{3\cdot 2^{k-2}-j-1} \text{ heaps.}$$

But m can be any one of the left leaves and the number of the left leaves is 2^{k-2}. For every value of m repeat the same steps; this will give us the result .

Claim 5: $A_2^{(j_1)} \cap A_2^{(j_2)} = \phi$ if $j_1 \neq j_2$ and $j_1, j_2 \in A_2$.

Proof: It is immediate from condition (3) in b) since if $j_1 < j_2$, j_1 represents a label for one of the right leaves for any x in $A_2^{(j_2)}$ and j_1 is a label for m for any x in $A_2^{(j_1)}$.

Claim 6: $A_2^{(j)} \cap A_1^{(i)} = \phi$ for any $i \in A_1$, $j \in A_2$.

Proof: This is clear from condition (3) in a and condition (3) in b.

Claim 7: There exists

$$\sum_{j \in A_2} |H_{2^{k-1}-1}|\, 2^{3\cdot 2^{k-2}-j-1}\, 2^{k-2}\, \binom{2^{k-2}-1}{j-2^{k-1}}(n-j)$$

heaps generating at most

$$\sum_{j \in A_2} |H_{2^{k-1}-1}|\, 2^{3\cdot 2^{k-2}-j-1}\, 2^{k-2}\, \binom{2^{k-2}-1}{j-2^{k-1}}$$

second order trees.

Proof: The result follows since the multiplicity for any second order tree generated from any x in A_2 is at least n-j.

c) Computation according to A_3.

For any $j \in A_3$ define $A_3^{(j)}$ as $A_3^{(j)} := \{x^{(j)};\ x^{(j)}$ is a heap with size n and $x^{(j)}$ satisfies the following conditions:

(1) the label of the node in m is j and its brother has a greater label than j;

(2) the labels of the internal nodes is the set $\{1,\ldots,2^{k-1}-1\}$;

(3) all the leaves except the node in m and its brother satisfy the condition that the label of the left leaves is greater than the label of its brother.

Claim 8: $|A_3^j|$ is at least $|H_{2^{k-1}-1}|\, 2^{k-2}(n-j)$.

Proof: We see from III.1.1 that the multiplicity for any second order tree generated from $x \in A_3^{(j)}$ is n-j. For every one of these heaps interchange the labels of the internal nodes. Since the number of heaps of

size $2^{k-1}-1$ is $|H_{2^{k-1}-1}|$, we get at least $(N-j)|H_{2^{k-1}-1}|$ heaps satisfying conditions (1), (2), and (3). Now m takes any node of the form $2^{k-1} + 2i$, $0 \leq i \leq 2^{k-2}-1$. Hence we have at least for any heap from the above, 2^{k-2} possible values for m. But for every value of m we have $(N-j)|H_{2^{k-1}-1}|$ heaps satisfying conditions (1), (2) and (3), then we have

$$|H_{2^{k-1}-1}|2^{k-2}(N-j) \text{ at least in } A_3^{(j)} .$$

Claim 9: $A_3^{(j_1)} \cap A_3^{(j_2)} = \phi$ for any $j_1 \neq j_2$ and $j_1, j_2 \in A_3$.

Proof: Immediate from condition (3) and (1).

Claim 10:

(α) $A_3^{(j)} \cap A_2^{(i)} = \phi$ for any $j \in A_3$ and $i \in A_2$;

(β) $A_3^{(j)} \cap A_1^{(i)} = \phi$ for any $j \in A_3$ and $i \in A_2$.

Proof: The result follows from combining conditions (3) in (a), (b) and (c).

Claim 11: There exists

$$\sum_{j \in A_3} |H_{2^{k-1}-1}| \ (N-j)2^{k-2} \text{ heaps}$$

generating

$$\sum_{j \in A_3} |H_{2^{k-1}-1}|2^{k-2}$$

second order trees. Combining all these results from Claim 3, 7 and 11, we obtain

$$|H_{2^k-1}| - f(k) \geq |J_{2^k-1}| . \quad \square$$

CHAPTER IV

Summary and Discussion

In this thesis we discussed several aspects related to the combinatorial properties of heapsort and we found the number of heaps with the maximum skewness of size N is equal to the number of heaps of size N-1 for any positive integer N. Also we proved in Chapter I that the number of heaps of size N the minimum skewness and for odd N is equal to $\frac{N!}{2\prod_{i=1}^{N} S_{i,N}}$ where $S_{i,N}$ is the size of the subtree rooted at node i. Combining these two relations with the relations given in Chapter II (II.3.2 and II.3.3), we found there exists $\frac{N!}{2\prod_{i=1}^{N} S_{i,N}} + \frac{(N-1)!}{\prod_{i=1}^{N-1} S_{i,N-1}}$ heaps of size N generate $2|J_{N-1}|$ second order trees of size N. The rest of the second order trees of size N are generated from $\left(\frac{N!}{2\prod_{i=1}^{N} S_{i,N}} - \frac{(N-1)!}{\prod_{i=1}^{N-1} S_{i,N-1}}\right)$.

Another relation we proved in Chapter I is the recursion formula (I.2.2). This recursion formula is valuable in calculating any number of heaps for any size with fixed conditions between any two offspring with the same parent.

In I.2 we proposed a new algorithm to generate the set of all heaps of size N from a subset of all heaps of size N-1. Comparing our algorithm with Williams' algorithm and Floyd's algorithm we mentioned that our algorithm saves space and time over these two known algorithms. In our algorithm, we construct a subset $<H_N^{B_m}$, of H_N, which serves as the base comparable to that of a vector space and from this set we can generate the set H_N. Therefore

in scheduling, if we use algorithm I.2, we can restrict our self to some element from $<H_N^{B_m}$ while if we use the other two algorithms, the heap generated must be an element from H_N.

Second order trees have been introduced and discussed in the present work. We proved that the exponential generating function of the number of this type of tree satisfies a nonlinear differential difference equation.

The numerical computation using the recursion formula in Chapter II.3.6 which gave us the number of the second order tree agreed with the results we got from the computer program which reflects the definition of the second order tree. The numerical computations in II.7 let us suggest the following conjecture:

for every positive integer N, there exists a constant C_N, $0.10 \leq C_N \leq 0.11$ such that the number of the second order trees of size N is $\lfloor (C_N+1)^N \rfloor$.

We prove in Chapter III in III.1.2 that number of the second order tree is smaller than $|H_N| - f(k)$, when $N = 2^k-1$.

From the above discussion, several interesting questions concerning the combinatorial properties are still open:

1. The complexity of our algorithm I.3.
2. Solution to the non-linear differential difference equations.
3. The weight of the second order trees.

REFERENCES

[AHU] A.V. Aho, J.E. Hopcroft, J.D. Ullman - The design and analysis of computer algorithms, Addison Wesley, (1974).

[AS1] A.V. Aho and N.J. Sloane - Some doubly exponential sequences, Fibonacci Quarterly 11 (4), 1973, 429-437.

[BEN] E.A. Bender - Asymptotic methods in enumeration, SIAM Review 16 (1974) 485-515.

[CAR] L. Carlitz - Some generalization of a binomial identity conjectured by Hogatt, The Fibonacci Quarterly, Vol. 19 #3, 1981, 200-208.

[DOB1] E.E. Doberkat - Inserting a new element into a heap. BIT-21 (1981) 255-269.

[DOB2] E.E. Doberkat - Deleting the root of a heap. Acta Informatica, 17 (1982) 245-265.

[DOB3] E.E. Doberkat - Some observation on the average performance of heapsort, 21st IEEE FOCS, Syracuse, NY 229-237, (1980).

[DOB4] E.E. Doberkat - An average case analysis of Floyd's algorithm to construct heaps. Information and Control (1984), in print.

[DOB5] E.E. Doberkat - Continuous models that are equivalent to randomness for the analysis of many sorting algorithms, Computing 31 (1983), 11-31.

[FLA] P. Flajolet - Combinatorial aspects of continued fractions, Discrete Math. 32 (1980) 125-161.

[FLO] R. Floyd - Algorithm 245 : Treesort 3, Comm. ACM 7 (1964), 701.

[GKu] D.H. Green and D.E. Knuth - Mathematics for analysis of algorithms, Birkhouser, Boston, (1981).

[GoJ] Ian P. Goulden and David M. Jackson - Combinatorial enumeration, John Wiley and Sons (1983).

[HEB] N. Habermann - Introduction to operating systems, Science Research Associates, Chicago 1976.

[KNU1] D.E. Knuth - The art of computer programming (Fundamental algorithms) Vol. 1, Addison Wesley, (1973) 2nd ed.

[KNU2] D.E. Knuth - The art of computer programming (sorting and searching), Vol. III, Addison Wesley (1973).

[LIU] C.L. Liu - Introduction to Combinatorial Mathematics, McGraw-Hill (1968).

[LOT] M. Lothaire - Combinatorices on words, Encyclopaedia of Mathematics, Vol. 17, Addison Wesley, (1983).

[LOV] L. Lovasz - Combinatorial problems and exercises - North Holland (1979).

[LOU] Louis Comtet - Advanced Combinatorics, D. Reidel Publishing Company (1974).

[McB] Elna B. McBirde - Obtaining generating functions, Springer-Verlag, Vol. I, (1971).

[MOO] J.W. Moon - Counting labeled trees, Canadian Mathematical Monograph, No. 1 (1970).

[ODL] A.M. Odlyzko - Periodic oscillations of coefficients of power series that satisfy functional equations, Bell Laboratories, Murray Hill, New Jersey.

[OLV] F.W.J. Olver - Asymptotics and special functions, Academic Press (1974).

[PSi] T. Porter, and I. Simon - Random insertion into a priority queue structure, IEEE Trans. Softw. Eng., SE1 (1975) 292-298.

[SED] R. Sedgewick - Mathematical analysis of combinatorial algorithms, Brown University, Technical report No. CS-82-09, Jan. (1982).

[VIL] N. Ta. Vilenkin - Combinatorics, Academic Press (1971).

[WIL] J.W.J. Williams - Algorithm 232 Heapsort, Comm. ACM 7, (1964) 347-348.

[ZIV] N. Ziviani - The fringe analysis of search trees, Dept. of Comp. Sci., University of Waterloo, Canada, Research Report CS-82-15, May 1982.

Symbols

A_N	is the set of all permutation of $\{1,\ldots,N\}$.
$B, B_m, B_{s'}, B_{s''}$	is some predicate sets over the set of all heaps of size N.
b(N,i)	is the brother for t(N,i).
C(N,ℓ)	is the number of sequences of size ℓ in $x \in H_N$.
$C_k(t)$	is the exponential generating function of $\{C(k,\ell)\}_\ell$.
H_N	is the set of all heaps of size N.
$H_N^{(max)}$	the set of all heaps of size N where the skewness is maximal.
J_N	the set of all second order tree generated from H_N.
$J_N^{(j)}$	is the set of all second order trees generated from H_N with fanout j at the root.
L_N	the set of all second order trees.
$L_N^{(i)}$	the set of all second order trees rooted at node i of $x \in H_N$.
$\mu(T_N)$	multiplicity of T_N.
$\nu(y)$	the number of heaps of size y.
SOT	second order tree.
t(N,i)	is the i-th special node.
T_N	tree generated from $x \quad H_N$.
$\lfloor x \rfloor$	is the greatest integer value less than or equal to x.
x[1..N]	array of real numbers represents a heap of size N.
x(i)	is the label for node i.