Relation Between the Lowest Level of "Sot" and the Total Weight

Ahmed Al-Jaber*

Received on Feb. 28, 2002 Accepted for publication on July 29, 2002

Abstract

In this article we will investigate an upper bound and lower bound of the number of operations needed to construct heap. This will be done by transforming the given heap into a complete weighted tree. We shall call this tree the second order tree. The numerical calculations are done to estimate the running time of the heap construction algorithm. We proved that heap construction is bounded below by constant time and bounded above by 2^((N+1)/2).

Keywords: heapsort, treesort, second order tree, sorting, algorithm, complexity.

Introduction:

A heap is an ordered, balanced binary tree in which the value at any node (key) is smaller than the values of its children. In other words heap can be viewed as an array of size N (i.e. X [1..N]) with the property that for any i, 2 ≤ i ≤ N, X [i] > X [i \ div 2]. Heap sort is a fairly complex algorithm used to sort data which pass through two different phases. The first phase is called the heap construction [1,11,19,20] and the second one is called the root deletion algorithm [2,3,8,14,16]. There are two important operations on heap sort (comparisons and swaps) where the running time of heap sort for these two operations is estimated to be N log_2 N [15,21].

Heap is used in many areas in computer science such as: in operating systems especially in scheduling [12]; in many graph algorithms such as shortest path problems [4,9,13,16,17]; in priority queues, and finally in many real world problems.

Several articles have been published to study the heap from different points of view for example algorithm design and algorithm complexity. Knuth [14], in his book “Art of Computer Programming”, gave the basic definitions and concepts

* Department Of Computer Science, King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan.
necessary to deal with heap such as walkdown concept, special path and two well
known Algorithms in which to construct heap (Williams algorithm and Floyds
involved the insertion of random element in a random heap. Dobrekat [7] gave the
running time of Floyds algorithm to construct heap. He proved that the running time
of this algorithm is $O(N \log_2 N)$. Finally, B. Bollobas [6] disucse both the best case
and the worst case for some types of the heap sort.

Construction of the second order tree.

Heap is a labeled balanced binary tree with the property that every son has a
greater label than its parent. In other words an array $X$ of size $N$ is a heap if-and only
if- for every index $i$, $2 \leq i \leq N$, $X[i] > X[i \div 2]$. The path leading from the index $N$
to the root is called a special path. Each node belongs to this path is called special
node, also the node having the same parent with the special node is called the brother
of this special node. If the brother of the special node is not defined then the label of
this brother is defined as an infinity value. The definition of the special node and it’s
brother is considered as an important factor of calculating the running time of any
algorithm in heap constructions and design. Therefore so one of the very important
definitions that can be deduced from the special node definition is the skewness of a
heap $X[1..N]$. It can be defined as the maximum number of swaps needed to insert a
new element $x$ into $X$ in order to generate a heap of size $N+1$ from $X[1..N]$.

Let $X[1..N]$ be a heap of size $N$. Let SK be the skewness of $X$. Define a new tree
from the heap $X$, so that the nodes of this tree represents the set of all inverse images
of $X$. After we apply Williams algorithm on each node, the nodes on the new tree
will generate $X$. We will denote this tree by SOT and it can be constructed by the
following algorithm:

Algorithm Second Order Tree

Input: Heap of size $N$
Output: Labeled second order tree

Begin.

Step1: Represent the root by $X$.

Step 2: Define the children of $X$ as the set of all heaps of size $N-1$ generated from
$X$. This set of children can generate $X$ after we insert the last key $x[N]$ in
$X[N-1]$. We will have $X[1..N]$ either by zero swap, one swap, ..., SK swap.
The children of the node $X$ can be organized from left to right depending on
the number of swaps needed to generate $X$. In other words from the
smallest number to the largest number of swaps needed.
Relation Between the Lowest Level of "Sot" and the Total Weight

**Step 3:** Recursively define the other levels until you reach the lowest level. It will contain all permutations which generate X[1, N] after we apply Williams algorithm to generate this heap.

**Step 4:** Start with the leftmost child of any node, label the edge leading to this child by $\emptyset$. From left to right, label the edge which leads from the right sibling of node $i$ to the root by $i+1$. END.

**Example:**

Let $X=1, 2, 4, 5, 3$


The root can be labeled by $x \Rightarrow X=12453$

Since $SK=2 \Rightarrow$ There are three children:

1. $12453$ generate $X$ by inserting $3$ in $1245$ with zero swap.
2. $13452$ generate $X$ by inserting $2$ in $1345$ with one swap.
3. $23451$ generate $X$ by inserting $1$ in $2345$ with two swaps.

After this step the SOT will look like:

```
  12453
   1  2
  12453  13452  23451
```

Repeat this process on each child of the first level (i.e. on $12453$, $13452$, $23451$). We will then get:

```
  12453
   0 1 2
  12453 15432 13452 35413 13451 25431 35411
```

841
Al-Jaber

Repeat the above process two more steps. We get:

Definition 1: Let $T_i$ be the SOT correspond to the heap $X_i$; Define $W(T_i)$ as the sum of the labels of all edges of $T_i$. Define $WL(T_i)$ as the sum of the labels of the lowest level of $T_i$.

Definition 2:

1) $W(T) = \sum_{T_1, T_2} W(T_i)$, where $H_N$ is the set of heaps of size $N$.

$W(T)$ is the set of possible swaps generated by Williams algorithm. The purpose is to construct the set of all heaps of size $N$ from the set of all possible input of size $N$ i.e $N!$ permutations.

Lemma: $WL(T) = \frac{N!}{2}$

Proof: Let $T_i$ be the SOT correspond to the heap $X_i$ of size $N$. All the leafs of $T_i$ represent the set of all possible inverse image to generate $X_i$. For all possible $i$'s the leaf is the set of all possible inputs which is $N!$ Inputs. Since every two children in the lowest level have the same parent one of them has a label zero while the other is one. Therefore every two children which have the same parent has a total weight of one. This implies that sum of the total weight of the lowest level of all possible second order tree is equal to $\frac{N!}{2}$.

Numerical Results and conclusion:

In order to estimate the average case analysis of Williams algorithm to generate heap, some of the computation has been done in table 1. It calculates several factors which are related to this purpose. One of these factors is the total number of the
weight \( W(T) \) and \( WL(T) \). All of these calculations have been done according to a given fixed size of the heap. The calculation in attribute 3 agrees with the results given by the lemma. One will notice from the comparison between attribute 2 and attribute 3 that for \( n > 3 \) the value of \( W(T) \) always can be smaller than the value of \( (N+1)! \). Both the computations on the table, as well as, the graph on page 6 show that the value of the ratio between \( WL \) and \( W \) become fixed after \( n = 7 \); In other words the 2.25 represent an upper bound.

So from both the lemma and the computation in table 1, we can estimate that the value of the average running time of Williams algorithm is always bounded below by

\[
1 = 2 \times \frac{N!}{2} / N! \quad \text{and bounded above by} \quad \text{O}(N) = 2 \times \frac{(N + 1)!}{N!} = 2(N+1)
\]

<table>
<thead>
<tr>
<th>Heap Size</th>
<th>Weight Grand Total</th>
<th>Lowest Level Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( W(T) )</td>
<td>Total WL(( T ))</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>125</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>786</td>
<td>360</td>
</tr>
<tr>
<td>7</td>
<td>5582</td>
<td>2520</td>
</tr>
<tr>
<td>8</td>
<td>38450</td>
<td>17136</td>
</tr>
<tr>
<td>9</td>
<td>280360</td>
<td>124960</td>
</tr>
</tbody>
</table>

Table 1: Exact value of \( W(T) \) and \( WL(T) \) Correspond to each heap size.

![Figure 1. Relation between heap size and WL(T)/W(T)](image-url)
العلاقة بين وزن المستوى الأخير والوزن الكلي للشجرة من الدرجة الثانية

أحمد الجابر

مناس

فننا في هذا البحث بالخصوص عن الحد الأعلى والحد الأدنى لعدد العمليات الضرورية لبناء الأكاداس المتعلقة بإخراج الفرز وذلك من خلال بناء شركة من المرتبة الثانية باستخدام شجرة المكدس ولقد أثبتت الحسابات الجدida تقاربًا كبيرًا ما بين النتائج النظرية والنتائج التجريبية.

References:

13. Horowitz, Ellis Sahni, Sartaj and Rajasekaran, Sanguthevar Computer
Relation Between the Lowest Level of "Sot" and the Total Weight

Algorithms / C++, Computer Science Pr 1996


17. Sedgewick, Rober and Flajolet, Philippe (Contributor), An Introduction to the Analysis of Algorithms, Addison-Wesley Pub Co 1996.


