DIMACS Technical Report 2001-30
September 2001

Scrambled Pairing and Square-Root Trees

by

Amr Elmasry
Computer Science Department, Rutgers University
elmasry@paul.rutgers.edu

ABSTRACT

An n-node forest of trees is called a square-root forest, if it has the following structure. For a given positive integer k the forest has $2k$ trees. The first $k+1$ are single nodes. For the other $k-1$ trees, the root of tree l has l single-node children, for all l from 1 to $k-1$. If $n \neq k + \frac{k\,(k+1)}{2}$ the definition is slightly different.

Given a forest of τ trees, the rank of a tree is defined to be the number of children of the root of this tree. A phase of operations is defined as first linking the trees in pairs, then replacing the tree with the largest rank with its sub-trees together with a new single-node tree. The pairing is done by first sorting the trees by rank and numbering them from 1 to τ, then linking the root of tree l to the root of tree $l + \lceil \tau/2 \rceil$, for all l from 1 to $\lfloor \tau/2 \rfloor$. We give a combinatorial proof that after applying an $O(n^{1.5})$ phases, the forest will converge to the square-root forest.

It is proven in [1] that using any pairing strategy, the amortized cost of deleting the item with the minimum value from a heap is $O(\sqrt{n})$. Our pairing strategy gives $\Theta(\sqrt{n})$ amortized cost for this operation.

Key-Words: Algorithms - Data Structures - Self-Adjusting Structures - Trees - Heaps.

1 Introduction

Given an n-node heap represented as a collection of trees, consider the following operations:

- insert: inserting a new item in the heap by adding a single-node tree to the collection of trees.
- deletemin: Removing the root node that has the smallest value and making each of its sub-trees a new tree in the heap.
- pair: Combining the trees of the heap in pairs by linking the root of one tree to another. To maintain the heap property, the root that has the larger value must be linked to the root that has the smaller value. The way to select pairs to be combined together matters, however.

We will combine the insert and deletemin operations in one operation and call it changemin. The cost of deletemin and pair operations is exactly the number of trees in the heap at the moment the operation is performed. A lower bound of $\Omega(\log n)$ for deletemin is inherited from sorting.

The pairing heap [1], a data structure that achieves $O(\log n)$ cost per operation in the amortized sense, is a self adjusting heap that uses pairing in its implementation. It is proven [1] that by altering the order of the trees before pairing, the amortized time of deletemin operation is $O(\sqrt{n})$.

In this paper, this upper bound is achieved, for any initial heap configuration (the number of trees in the heap will be $\Theta(\sqrt{n})$ in the steady state), by applying a sequence of interleaved pair and changemin operations with some restrictions on the linking and pairing strategies. A pairing phase followed by a changemin operation will be called a step. More interesting is that after an $O(n^{1.5})$ steps any heap will converge to the same structure. We call this structure the square-root forest.

In section 2, the motivation behind our pairing strategy is discussed. In section 3, the rules for pairing, linking, and choosing the node with the minimum value are stated. The main theorem is proved in section 4. Section 5 deals with the case when $n \neq k + \frac{k\,(k+1)}{2}$. And in section 6, we construct an initial forest that requires $\Theta(n^{1.5})$ steps to converge to the square-root forest.

2 Pairing

Define the rank of a node to be the number of children of this node. Define the rank of a tree to be the rank of the root of this tree. Let r_u be the rank of tree u. For a given choice of which trees are to be paired together, taking the summation over all combined pairs of trees u and v. Let

$$J = \sum \frac{min(r_u, r_v)}{(r_u + r_v)}$$

Intuitively, combining trees close in rank helps in minimizing the number of trees of the heap in the long run. A good strategy to maximize the number of trees in the long run is to select the pairs of trees that will be combined together in a way to minimize J.

Theorem 1 *Let τ be the number of trees of the heap. Having the sequence of trees sorted by rank and numbered from 1 to τ, to minimize J, tree l is to be combined with tree $l + \lceil \tau/2 \rceil$, for all l from 1 to $\lfloor \tau/2 \rfloor$.*

Proof: Consider the case of having 4 trees T_1, T_2, T_3, T_4 with ranks r_1, r_2, r_3, r_4 respectively, where $r_1 \leq r_2 \leq r_3 \leq r_4$. There are 3 ways to pair them:

Pairing 1 : Combining T_1 with T_2 and T_3 with T_4.

Pairing 2 : Combining T_1 with T_3 and T_2 with T_4.

Pairing 3 : Combining T_1 with T_4 and T_2 with T_3.

It can be shown that

$$\frac{r_1}{r_1 + r_3} + \frac{r_2}{r_2 + r_4} \leq min(\frac{r_1}{r_1 + r_2} + \frac{r_3}{r_3 + r_4}, \frac{r_1}{r_1 + r_4} + \frac{r_2}{r_2 + r_3})$$

Therefore pairing 2 is the one that minimizes J.

For the general case of τ trees, assume that after sorting the trees by rank a tree u was combined with a tree v, where $r_u < r_v$ and $v \neq u + \lceil \tau/2 \rceil$. There must exist another two combined trees x and y, where $r_x < r_y$, such that one of the following holds:

$r_x > r_u$ and $r_y < r_v$ (pairing 3).

$r_x < r_u$ and $r_y > r_v$ (pairing 3).

$r_x < r_u$ and $r_y < r_v$ (pairing 1).

$r_x > r_u$ and $r_y > r_v$ (pairing 1).

In all cases, the pairing of these four trees can be changed to get a smaller or equal value for J (pairing 2), a contradiction. $\square$

3 Reaching the Bound

Define the rank of the heap to be the largest rank of a tree in the heap. Let τ be the number of trees of the heap. To maximize the amortized cost of deletemin operation, the following three rules are imposed:

- The trees are sorted by rank and numbered from 1 to τ. The pairing is done by combining tree l with tree $l + \lceil \tau/2 \rceil$, for all l from 1 to $\lfloor \tau/2 \rfloor$.
- When two trees are combined, the tree with the smaller rank is linked to the tree with the larger rank.
- For a deletemin operation, the minimum value is assumed to be at the root of the tree with the largest rank.

4 The Square-Root Forest

Assume that the number of nodes in the forest is $n = k + \frac{k\,(k+1)}{2}$ for a given positive integer k. We call the forest a square-root forest when it has $2k$ trees, $k+1$ of them are single nodes and for the other $k-1$ trees the root of tree l has l single-node children, for all l from 1 to $k-1$. See Figure 1. If n does not satisfy the above equation the definition is slightly different and will be considered in section 5.

Theorem 2 *For any initial n-node forest of priority queues, applying a sequence of interleaved pairing & changemin phases and following the above three rules, the heap will converge to a square-root forest after an $O(n^{1.5})$ steps.*

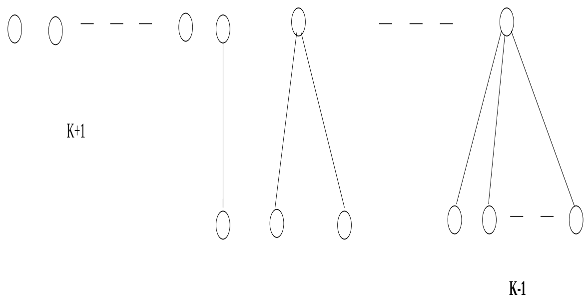

Figure 1: The square-root forest

Let τ_i be the number of trees in the heap after step i,
c_i be the rank of the heap after step i.

After the pairing phase of step $i+1$, the number of trees will be $\lceil \tau_i/2 \rceil$. The deletemin operation replaces the tree with the largest rank with its $c_i + 1$ subtrees. Finally a new single-node tree is inserted. After step $i+1$, the number of trees of the heap will be

$$\tau_{i+1} = \lceil \tau_i/2 \rceil + c_i + 1 \tag{1}$$

Define η_i and β_i such that

$$c_i = \lfloor \tau_i/2 \rfloor + \eta_i \tag{2}$$

$$c_{i+1} = c_i + 1 + \beta_{i+1} \tag{3}$$

Using (1) and (2)

$$\tau_{i+1} = \tau_i + \eta_i + 1 \tag{4}$$

Using (1) and (3)

$$c_{i+1} = \tau_{i+1} - \lceil \tau_i/2 \rceil + \beta_{i+1} \tag{5}$$

Using (4) and (5)

$$c_{i+1} = \begin{cases} \lfloor \tau_{i+1}/2 \rfloor + \lceil \frac{\eta_i}{2} \rceil + \beta_{i+1} & \text{if } \tau_i \text{ is odd} \\ \lfloor \tau_{i+1}/2 \rfloor + \lceil \frac{\eta_i+1}{2} \rceil + \beta_{i+1} & \text{if } \tau_i \text{ is even} \end{cases}$$

$$\eta_{i+1} = \begin{cases} \lceil \frac{\eta_i}{2} \rceil + \beta_{i+1} & \text{if } \tau_i \text{ is odd} \\ \lceil \frac{\eta_i+1}{2} \rceil + \beta_{i+1} & \text{if } \tau_i \text{ is even} \end{cases} \tag{6}$$

The state of the heap after step i is defined in terms of the value of η_i. Performing step i corresponds to a transition from one heap state to another, that depends solely on the value of β_i.

Lemma 1 *The total number of transitions with a positive value of β_i is $O(n)$ and $\sum_{i:\beta_i>0} \beta_i$ is $O(n)$.*

Proof: Because of our pairing strategy, when a node is linked to another, it must have a smaller or equal rank. A positive β_i is a result of a sub-tree which has at least two children more than the number of children of its father. These sub-trees may exist in the initial heap configuration, and at least one of them is cut from its father with each of these transitions. $\square$

Define the following heap states. See Figure 2 for the state transitions. (Note that the transitions with positive $\beta_i s$ are not considered. From this point on we will ignore these transitions as they will not affect the bounds driven)

1. $\eta_i \leq -3$, or $\eta_i = -2$ and τ_i is odd:
   Using (4) then $\tau_{i+1} \leq \tau_i - 1$.
   Using (6) then $\eta_{i+1} \leq -1$.

2. $\eta_i = -2$ and τ_i is even:
   Using (4) then $\tau_{i+1} = \tau_i - 1$.
   Using (6) then $\eta_{i+1} \leq 0$.

3. $\eta_i = -1$:
   Using (4) then $\tau_{i+1} = \tau_i$.
   Using (6) then $\eta_{i+1} \leq 0$.

4. $\eta_i = 0$ and τ_i is odd:
   Using (4) then $\tau_{i+1} = \tau_i + 1$.
   Using (6) then $\eta_{i+1} \leq 0$.

5. $\eta_i = 0$ and τ_i is even:
   Using (4) then $\tau_{i+1} = \tau_i + 1$.
   Using (6) then $\eta_{i+1} \leq 1$.

6. $\eta_i = 1$:
   Using (4) then $\tau_{i+1} = \tau_i + 2$.
   Using (6) then $\eta_{i+1} \leq 1$.

7. $\eta_i \geq 2$:

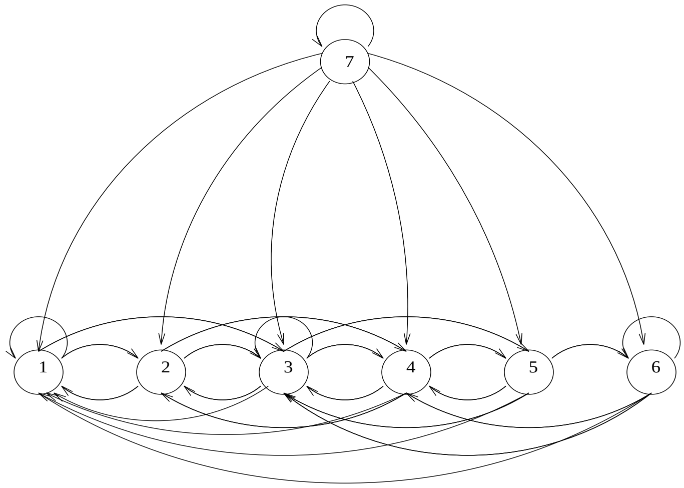

Figure 2: The state transitions

We call states 1, 5, 6 and 7 t-states and states 2, 3 and 4 s-states.

Let N^j be the total number of visits to state j, and s_i be the state entered after step i.

Lemma 2 *The total number of visits to state* 7 *is* $O(n)$ *and* $\sum_{i:s_i=N^7} \eta_i$ *is* $O(n)$.

Proof: Using (6)

$$\eta_{i+1} \leq \begin{cases} \frac{\eta_i}{2} + 1 + \beta_{i+1} & \text{if } \tau_i \text{ and } \eta_i \text{ are even} \\ \frac{\eta_i}{2} + \frac{1}{2} + \beta_{i+1} & \text{otherwise} \end{cases}$$

Using (4), the parity of the number of trees in the heap changes if η_i is even, the following relation follows

$$\eta_i \leq \frac{\eta_{i-2}}{4} + \frac{5}{4} + \beta_i + \frac{\beta_{i-1}}{2}$$

Applying the above relation recursively, the value of η_i can be defined in terms of the initial η_0

$$\eta_i < \frac{\eta_0}{2^i} + \frac{5}{3} + \sum_{l=1}^{i} \frac{\beta_l}{2^{i-l}}$$

Taking the summation over all the visits to state 7 and only considering positive β_is

$$\sum_{i:s_i=7} \eta_i < |2\eta_0| + \frac{5}{3}N^7 + \sum_{i:\beta_i>0} 2\beta_i$$

Using Lemma 1

$$\sum_{i:s_i=7} \eta_i < \frac{5}{3}N^7 + O(n) \tag{7}$$

When the heap enters state 7 at any step i, the relation $\eta_i \geq 2$ holds, and hence

$$\sum_{i:s_i=7} \eta_i \geq 2N^7 \tag{8}$$

Using (7) and (8), the lemma follows. □

Define a bad node to be any node that has a positive rank and is not a root of a tree. A bad sub-tree is a sub-tree whose root is a bad node (if a sub-tree is not a single node then it is a bad sub-tree).

Lemma 3 *The total number of visits to state* 5 *and to state* 6 *is* $O(n)$.

Proof: Let $R^{i,j}$ be the transition from state i to state j. For the following transitions: $R^{3,4}$,$R^{3,5}$,$R^{4,5}$,$R^{5,6}$,$R^{6,6}$, β_i must be 0. We call these transitions hard or h-transitions. An h-transition happens either if the rank of a node exceeds the rank of its father by one (a case which happens only an $O(n)$ times as a result of the initial heap configuration), or if two trees have the same positive rank. For two consecutive h-transitions to happen (other than the $O(n)$ that may happen as a result of the initial configuration) three of the heap's trees have to be created with the same positive rank.

A sequence of transitions starting and ending at the same state and not visiting it in between is called a simple loop at this state.

Let δ_i be the number of nodes whose ranks become positive at step i; a case which happens only when a single-node tree is linked to another. The number of trees with positive ranks after step $i+1$ is at least $\lfloor \tau_i/2 \rfloor - 1$. The number of single-node trees is at most $c_i + 3$. (In the pairing phase at step $i+1$, a new tree is added to the c_i children of the tree with the largest rank. In the chagemin operation, a new node is inserted. And, a third node may be a leftover if τ_i is odd). Using (2)

$$\delta_{i+1} \leq \begin{cases} \lfloor \frac{\eta_i+4}{2} \rfloor & \text{if } \tau_i \text{ is odd} \\ \lfloor \frac{\eta_i+3}{2} \rfloor & \text{if } \tau_i \text{ is even} \end{cases}$$

The relation $\delta_{i+1} \geq 3$ holds only if $\eta_i \geq 2$; a condition that is true at state 7 only. The total number of a second-in-a-row h-transitions is $\sum_{i:\delta_i \geq 3}(\delta_i - 2) + O(n)$

which is $O(n)$ by Lemma 2. From the state transitions, any state 6 simple loop must involve a second-in-a-row h-transition. The total number of visits to state 6 is therefore $O(n)$.

Consider the simple loop, at state 5, that involves the transition from state 5 to state 4 and back to state 5. For these transitions to happen, there must exist a pair of trees with the same positive rank, say r, and another pair of trees with the same rank $r - 1$. There are $O(n)$ such trees as a result of the initial heap configuration. This sequence of trees can only be produced by entering state 6. Hence, these simple loops can happen an $O(n)$ times only.

Consider the simple loop, at state 5, that involves the transitions from state 5 to state 4, to state 3, and back to state 5. For these transitions to happen, there must exist a pair of trees with the same positive rank, say r, a tree with rank $r-1$, and another pair of trees with the same rank $r - 2$. There are $O(n)$ such trees as a result of the initial heap configuration. This sequence of trees can only be produced as a result of the simple loop at state 6 to state 5 and back to state 6. Hence, these simple loops can also happen an $O(n)$ times only.

The relation $\delta_i = 2$ holds only at state 4, state 6, or state 7. The number of h-transitions is $\sum_{i:\delta_i \geq 2}(\delta_i - 1) + O(n)$ which is bounded from above by $N^4 + O(n)$.

From the state transitions, for any other state 5 simple loop, the number of h-transitions involved is greater than the number of visits to state 4. The total number of visits to state 5 is, therefore, $O(n)$.

Some trees with positive ranks may be linked to other trees and produce bad nodes. When bad nodes become trees again, in a later step, there may exist other trees with the same ranks, and this may produce an h-transition. Fortunately, this will not affect the above analysis. Assume that at step i_1 some trees with positive ranks are linked to other trees, and at step i_2 one of these bad sub-trees became a tree again. Assume that there was another tree that has the same rank as this sub-tree at this moment. This other tree must have been created at a step i_3, where $i_1 < i_3 < i_2$. For every step from i_1 to i_2, including step i_3, at least one bad sub-tree must have become a tree. This means that after step $i_3 - 1$ the heap was at either state 4, state 6 (with $\delta_i \leq 1$), or at state 7. Hence, an h-transition was over-counted at step i_3. □

Lemma 4 *The total number of visits to state* 1 *is* $O(n)$ *and* $\sum_{i:s_i=1} -\eta_i$ *is* $O(n)$.

Proof: Applying (3) for all steps from 1 to t, the sum telescopes

$$\sum_{i=1}^{t}(\beta_i + 1) = c_t - c_0 = O(n) \tag{9}$$

Similarly using (4)

$$\sum_{i=1}^{t} -(\eta_i + 1) = \tau_1 - \tau_{t+1} = O(n) \tag{10}$$

Adding (9) and (10)

$$\sum_{i=1}^{t} -(\eta_i - \beta_i) = O(n)$$

Define f_i such that

$$f_i = \begin{cases} \lceil \frac{\eta_i}{2} \rceil & \text{if } \tau_i \text{ is odd} \\ \lceil \frac{\eta_i+1}{2} \rceil & \text{if } \tau_i \text{ is even} \end{cases}$$

Using (6)

$$\sum_{i=0}^{t-1} -f_i = O(n) \tag{11}$$

Using the state definitions, The following relations hold after any step i

$$\begin{array}{ll} f_i = 1 & \text{at state 5, or state 6.} \\ f_i = 0 & \text{at state 2, state 3, or state 4.} \\ f_i \leq -1 & \text{at state 1.} \end{array}$$

And using Lemma 2

$$\sum_{i:s_i=N^7} f_i = O(n)$$

Substituting in (11)

$$N^1 \leq \sum_{i:s_i=1} -f_i = N^5 + N^6 + O(n)$$

Using Lemma 3, the lemma follows

□

Let b_i be the number of bad nodes of the heap after the pairing phase at step $i+1$. The total decreases in the number of bad nodes is therefore $\sum_{i:b_{i+1}-b_i<0}(b_i - b_{i+1})$.

Lemma 5 *The total decreases in the number of bad nodes is $O(n)$.*

Proof: At step $i+1$, before the changemin operation, the number of trees of the heap is $\lceil \tau_i/2 \rceil$ and its rank is c_i+1. The goal is to find an upper bound on $b_{i+1}-b_i$. This bound is achieved when all the trees have positive ranks, while all the sub-trees of the tree with the largest rank are single nodes. Hence, after the changemin operation, there will be $\lceil \tau_i/2 \rceil - 1$ trees with positive ranks and c_i+2 single-node trees. The increase in the number of bad nodes at the next pairing phase will be the floor of half the difference between the number of trees with positive ranks and the number of single node trees.
Therefore, the relation $b_{i+1}-b_i \leq \lfloor \frac{\lceil \tau_i/2 \rceil - c_i - 3}{2} \rfloor$ holds. Using (2)

$$b_{i+1}-b_i \leq \begin{cases} \lfloor \frac{-\eta_i-2}{2} \rfloor & \text{if } \tau_i \text{ is odd} \\ \lfloor \frac{-\eta_i-3}{2} \rfloor & \text{if } \tau_i \text{ is even} \end{cases}$$

Using the state definitions, $b_{i+1}-b_i$ may be positive at state 1 only. Using this fact together with Lemma 4

$$\sum_{i:b_{i+1}-b_i>0} (b_{i+1}-b_i) < \sum_{i:s_i=1} -\eta i = O(n)$$

For any step t

$$\sum_{i:b_{i+1}-b_i<0} (b_i-b_{i+1}) = \sum_i (b_i-b_{i+1}) + \sum_{i:b_{i+1}-b_i>0} (b_{i+1}-b_i)$$

$$= b_0 - b_t + O(n) = O(n)$$

□

Lemma 6 *After spending $O(\sqrt{n})$ steps in the s-states, either*

- *The number of the bad nodes decreases, or*
- *The heap visits a t-state, or*
- *The heap reaches the steady state and never leaves the s-states.*

Proof: Consider the case when the heap was at state 3 or state 4 after step i, and at least one of the sub-trees of the tree with the largest rank was a bad sub-tree after the pairing phase of step $i+1$. One of these bad sub-trees will become a tree after the pairing phase of step $i+2$, and the number of the bad nodes decreases. If the heap was at state 2 after step i, then this bad sub-tree will still be a sub-tree

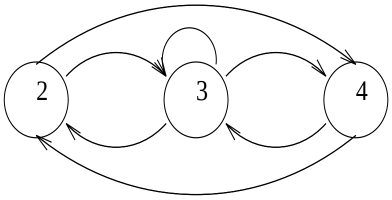

Figure 3: The steady state transitions

at the next step, and it will again be a sub-tree of the tree with the largest rank and the heap will be at either state 3 or state 4.

Assume that the heap stays at the s-states for a sequence of w consecutive steps and that all the children of the root of the tree with the largest rank are single nodes for each of these steps. Consider the node that was inserted when the heap first enters the s-states. This node will always be the root of a tree and its rank increases with every step. If the heap enters the s-states at state 2, start counting from the step after.

Assume that the rank of the heap was q when it entered the s-states. When entering state 4 the rank of the heap increases by 1. When entering state 2 the rank of the heap decreases by 1. When entering state 3 the rank of the heap does not change. Every time state 2 is entered state 4 must be entered before coming back to state 2 and vice versa. The rank of the heap will therefore be at least $q-1$ and at most $q+1$ after each of these w consecutive steps. See Figure 3.

If $w \geq q+2$, then the tree which was inserted when the heap first entered the s-states must have been deleted within these w steps and the heap must have reached the steady state.
If $w < q+2$, then before these w steps, the heap must have had at least $w-2$ nodes where the rank of node l is greater or equal to $q-l$, for all l from 1 to $w-2$. But $\sum_{l=1}^{w-2}(q-l) \leq n$ and $w < q+2$ implies $w = O(\sqrt{n})$. $\square$

Corollary 1 *For every step i after an $O(n^{1.5})$ steps, either*

- $c_i = \lfloor \tau_i/2 \rfloor - 2$ *and τ_i is even, or*
- $c_i = \lfloor \tau_i/2 \rfloor - 1$, *or*
- $c_i = \lfloor \tau_i/2 \rfloor$ *and τ_i is odd.*

Moreover, all the sub-trees will be single nodes.

Proof of Theorem 2: To enter state 4 two trees with the same positive rank must exist. To enter state 2 a tree with a positive rank must have been missing from the definition of the square-root forest. At the steady state, the heap must visit state 2 every time it visits state 4, before returning back to state 4. The total number of nodes must then be $\frac{m\,(m+1)}{2} + l$, where $m + 1 \leq l \leq 2m$, for some integers m and l. Given that $n = k + \frac{k\,(k+1)}{2}$ for a given integer k, there are no integer solutions for l in terms of k. We conclude that the heap will never leave state 3. The theorem follows. □

5 General Configurations

For the case when $n = k + \frac{k\,(k+1)}{2} + m$, where $m \leq k+1$ for any positive integers m and k, it can be shown that the steady state cycle will consist of $k+1$ configurations. All the configurations are similar to the square-root forest with a little variation. The first $k + 2 - m$ configurations have $2k + 1$ trees. They differ from the square-root forest by having: an extra single-node tree, and for all l from 1 to $k + 2 - m$ there is an extra tree of rank $m + l - 2$ and a missing tree of rank $l - 1$. The other $m - 1$ configurations have $2k + 2$ trees. They differ from the square-root forest by having: an extra single-node tree, a new tree of rank k, and for all l from 1 to $m - 1$ there is an extra tree of rank $l - 1$ and a missing tree of rank $k + l + 1 - m$. Note that if $l = m - 1$ then $k + l + 2 - m$ equals $k + 1$.

6 Constructing an Initial Configuration

There are some initial forests that need $\Theta(n^{1.5})$ to converge to the square-root forest. An initial configuration that needs $\Theta(n^{1.5})$ to converge to the square-root forest is constructed as follows: Begin by using a positive proportion of the nodes, say around half the nodes, and construct a typical square-root forest. Construct a linear chain of nodes (every node has one child except for the only leaf). Add this chain as a child of any other tree in the forest. The idea behind this construction is having $\Theta(n)$ bad nodes. There is only one tree that has all the bad nodes. Every time it takes $\Theta(\sqrt{n})$ steps for the linear chain to gain $\Theta(\sqrt{n})$ children and become the tree with the largest rank, and the number of bad nodes decreases by one.

References

[1] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. *The Pairing heap: a new form of self_adjusting heap.* Algorithmica 1,1 (1986), pp. 111-129.