DIMACS Technical Report 97-52
September 1997

Pairing Heaps are Sub-optimal

by

Michael L. Fredman[1]
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903

[1]Permanent Member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs, Bellcore, and Bell Labs.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

Pairing heaps were introduced as a self-adjusting alternative to Fibonacci heaps. They provably enjoy $\log n$ amortized costs for the standard heap operations. Although it has not been verified that pairing heaps perform the decrease key operation in constant amortized time, this has been conjectured and extensive experimental evidence supports this conjecture. Moreover, pairing heaps have been observed to be superior to Fibonacci heaps in practice. However, as demonstrated in this paper, pairing heaps do not accommodate decrease key operations in constant amortized time.

1 Introduction

Pairing heaps were introduced [1] as a self-adjusting alternative to Fibonacci heaps [2]. They are easy to code and provably enjoy $\log n$ amortized costs for the standard heap operations. Although it had not been verified that pairing heaps perform the decrease key operation in constant amortized time (the raison d'etre of Fibonacci heaps), this has been conjectured [1] and extensive experimental evidence [3, 5] supports this conjecture. These same experimental studies suggest that pairing heaps are superior to Fibonacci heaps in practice. The good observed behavior of pairing heaps has led to their wide-spread use; they are included, for example, as one of several implementations of priority queues in the Gnu C++ library. However, as demonstrated in this paper, pairing heaps do not accommodate decrease key operations in constant amortized time.

The conceptual simplicity of pairing heaps justify their occasional inclusion in the undergraduate algorithms curriculum [4, 6]. Although simple in terms of their design, their behavior may ultimately prove to be complex.

2 Pairing Heaps

We begin with a brief description of the pairing heap data structure. The pairing heap uses a tree structure to store the values in the heap, one value per tree node. The placement of stored values respects the *heap order* condition: the value stored in the parent of a node is $\leq$ the value stored in the node itself. We thus find the minimum heap value stored in the tree root.

The heap operations for a pairing heap are implemented as a series of *linking operations*. Let H_1 and H_2 be heap ordered trees with respective roots x_1 and x_2, and respective root values v_1 and v_2. A linking operation applied to the pair (x_1, x_2) inserts H_1 into H_2 as the leftmost subtree of H_2 if $v_1 \geq v_2$, and otherwise inserts H_2 into H_1 as the leftmost subtree of H_1. To perform an insertion of a value v into a heap whose tree is H, a single node tree containing v is linked with the root of H. To perform a decrease operation on the value v stored in a node x of H which is not the root of H, we first remove the subtree of x from H by removing the link from x to its parent in H. Next, we link x with the root of H, having first decreased the value v in x. (If x is the root of H, we directly decrease its value.) To perform a deletemin operation, we remove the root of H and proceed to perform linkings among the children of this root in a certain prescribed order until a single root is restored. The order of the linkings is as follows. Let $x_1, \cdots, x_k$ be the children of the root in left-to-right order. We start by linking the pairs (x_1, x_2), $(x_3, x_4) \cdots$. Let $y_1, \cdots, y_h$, $h = \lceil k/2 \rceil$, be the surviving roots. (If k is odd, then $y_{\lceil k/2 \rceil}$ is x_k.) We finish by linking the pair (y_{h-1}, y_h), then linking y_{h-2} with the root that results from the preceding linking etc., until finally we link y_1 with the root of the structure formed from the linkings of $y_2, \cdots, y_h$.

Several variants of this data structure have been suggested [1, 5].

The significant and interesting feature of pairing heaps, as seen from our description, is that the selection of operands for the various node pairings does not take into consideration structural aspects of the respective trees (e.g. tree size), unlike the manipulations of Fibonacci heaps. As a consequence, the pairing heap is considered to be a *self-adjusting* data structure. However, as this paper demonstrates, this self-adjusting aspect of pairing heaps precludes optimal performance of the heap operations.

3 Analysis

O-sequences

Our lower bound is established by considering sequences of operations structured as follows. First, n insertions are performed so that n items are present in the heap. The remaining operations are partitioned into *rounds*. Each round begins with $L = \lceil \log_3 n \rceil$ decrease key operations performed on children of the root and concludes with an insertion, followed by deletemin. Upon its conclusion, the execution of a round leaves unchanged the number of items n in the heap. Operation sequences structured in this manner are referred to as *O-sequences*.

If pairing heaps enjoyed constant amortized time decrease key costs, then the total amortized cost of executing a given round of an O-sequence would be $O(\log_3 n)$, and the total execution cost of an O-sequence of length $m \geq n$ would be bounded by $O(m + n \log_3 n)$. We will see that this is not the case.

Our analysis considers the expected execution cost of an O-sequence relative to a random variable X over these sequences to be defined. The node linkings that take place during the execution of operations are to be governed by an adversary defined in the sequel. One consequence of this adversary is that the decrease key operations taking place in a given round permute the children of the root, and an insertion adds a new node as the leftmost child of the root. Moreover, for a tree of size n, the root must have at least $\lceil \log_3 n \rceil$ children. The lower bound argument to be developed captures the following intuition. To achieve the execution of deletemin operations with $O(\log_3 n)$ work requires that the various node linkings that take place for the most part be *efficient*, meaning that the sizes of subtrees being linked are somewhat commensurate. But the disruptive effect of the decrease operations, which scramble the children of the root, force most linkings to be inefficient.

We proceed to define the random variable X over the O-sequences of length m. Let $C = C(n)$ be a constant depending on n such that $C \geq 1$. X is constructed by independently choosing for each round a random sequence σ consisting of L distinct integers in $[1, C \cdot L]$. Each sequence σ is chosen with uniform probability from among the set of possible sequences. Having chosen $\sigma = \sigma_1, \cdots, \sigma_L$, the j^{th} decrease key operation of the round is performed on the σ_j^{th} child of the root (as the tree exists at the beginning of the round and counting from the left). Should there not be as many as σ_j children, then the j^{th} decrease key operation is vacuous.

Adversary

Our adversary for linking nodes is defined in terms of a rank function assigning integer values to the nodes. For nodes v and w with respective ranks r_v and r_w, a linking of v and w, v to the left of w, will place v in the root position if and only if $r_v > r_w$. The rank function requires a non-negative parameter $d = d(n)$ chosen in a manner depending on n. A node with no children has rank 0. If v has rank k and a new child w is linked to v as its leftmost child, then v's rank is incremented provided that w has rank $\geq k-d$. We refer to this linking as an *efficient* linking, and we refer to the link joining the child to the parent as an *e-link*. Now, if subsequent to its last change of rank there have been 3^d children added to v with rank $< k-d$ (so that v's rank was not incremented with the addition of these children), then the next child w to be added causes v's rank to be incremented, irrespective of w's rank. A linking is called *incremental* if it causes the rank of the parent node to be incremented. An incremental linking which is not efficient is referred to as a *default* incremental linking. The adversary will also obey the convention that the various decrease key and insertion operations performed will result in the affected node becoming a child of the current root. In particular, a node retains its same parent upon undergoing a decrease key operation in the course of executing a given O-sequence.

Some properties of our adversary are embodied in the following lemmas. They assume that trees have been formed by linking in accordance with the adversary. In the sequel, the rank of a tree is defined to be the rank of its root.

Lemma 1 *A tree of rank k has at most 3^k nodes.*

Proof: Let s_k denote the maximum possible size of a tree of rank k. This tree is formed by starting with the maximum size tree of rank $k-1$, adding (with an incremental link) a maximal size child of rank $k-1$, and then, assuming $k \geq d+1$, adding 3^d maximal size children of rank $k-d-1$ (this being the maximum allowable rank for non-incremental links). Thus we obtain $s_k \leq 2s_{k-1} + 3^d s_{k-d-1}$. The conclusion follows by induction. □

Lemma 2 *Suppose T is a tree with $> 3^{L-1}$ nodes and that its root r has at most $C \cdot L$ children, and d satisfies $C \leq (3^d+1)/2$. Then r has at least $L/2$ children with e-links to r, each having rank $< L$.*

Proof: By lemma 1 r has rank at $\geq L$. Let u be the number of default incremental links connected to r, so that the number of e-links that connect r to children with rank $< L$ is at least $L-u$. Each default incremental link accounts for 3^d+1 children of r (the link itself and the immediately preceding 3^d non-incremental links). Thus $u \leq C \cdot L/(3^d+1) \leq L/2$. The lemma follows immediately. □

Lemma 3 *Among the children with e-links to r at most $d+1$ that have a common rank value.*

Proof: This follows as a consequence of the facts that each e-link increases the rank of its parent, and the rank of the child is within d of the rank of the parent before the child was acquired. □

Schedule of Obligation

Our lower bound analysis only takes into consideration linkings that actually take place during deletemin operations, omitting the linkings associated with decrease key and insertion operations. An *efficient event* for a node is said to take place either when the node becomes a child of another node with an efficient linking, or when the node gets deleted in a deletemin operation. A node can have a series of efficient events during its life in the heap which culminates with its deletion in a deletemin operation (its final efficient event). We associate with each node in the heap a *schedule of obligation*. Coming into effect at the onset of a given round, the schedule specifies for each possible continuation of the O-sequence a quantity ℓ which usually represents the number of linking operations involving that node up to (and including) its next efficient event. After a node has its next efficient event, at which point it has satisfied the obligatory number of linkings specified in the appropriate entry of its schedule, then assuming the node has not been deleted, its schedule of obligation gets renewed at the onset of the round in which its parent gets deleted from the heap. The schedule of obligation of a given node may get repeatedly renewed over the span of time before the node is deleted from the heap. Each renewal obligates that node to additional linkings, as specified by the schedule.

At the conclusion of the O-sequence, some of the remaining nodes in the heap will likely not have e-links to their parents. For the convenience of our analysis, these nodes are given *stipulated* obligations in the corresponding entries of their respective schedules of obligation even though the linkings specified by these stipulated obligations are likely not to have taken place. At the conclusion of the O-sequence, the total number of linkings that have taken place is at least one-half the amount of total obligation that has been satisfied (one linking involves two nodes). Some unsatisfied obligation may exist among nodes residing in the heap at the conclusion of the O-sequence, reflected by the stipulated obligations.

Extraction of Ternary Decision Tree

At the onset of a given round of an O-sequence, let $v_1, \cdots, v_p$ be nodes with e-links to the root and with ranks $< L$. As discussed above, these nodes receive renewed schedules of obligation. Fix a continuation of the O-sequence beyond the given round, so that variation exists only with the choice of the decrease key operations within the round. Furthermore, fix the randomly selected of set of L nodes to participate in the decrease key operations of the round, and assume that these p nodes v_i are included among these choices. Finally fix the placement of the other $L - p$ selected nodes in the sequence of the decrease key operations, so that remaining variation exists only in the choosing of one of the $p!$ possible placements of the v_i nodes within their fixed set of p positions as children of the root. Number these p particular positions from 1 to p, and let U_i be a variable whose value is the node v_j arriving in position i. Let R designate the restrictions that have been imposed. Given R and a specific assignment of the v_i's to the variables U_i, the O-sequence continuation is now completely specified. Consider the moment that the next efficient event takes place for one of these v_i. Subsequent to this moment we say that v_i has *become efficient*, and up to this moment we say that v_i is *pre-efficient*. The number of linkings involving v_i while it is pre-efficient represents its renewed obligation for the specified O-sequence continuation. In the sequel we bound the total expected obligation of the v_i's, averaging over the $p!$ possible assignments to the U_i's.

We need the following notation: Let x_1 and x_2 be heap nodes and assume that neither is an ancestor of the other. The expression, link(x_1,x_2), designates the operation that links x_1 to x_2 with x_1 as the leftmost child of x_2. The expression blink(x_1,x_2) represents a binary decision for the linking of x_1 and x_2; the result is either link(x_1,x_2) or link(x_2,x_1).

Assume restrictions R are in effect. For the moment we drop the understanding that our adversary is to govern the linkings that take place, but retain the convention that the insertions and decrease key operations performed will result in the affected node becoming a child of the current root. Given a specific assignment α of the v_i's to the variables U_i, the various linkings that take place over the course of the subsequent deletemin operations of the O-sequence can be modeled as a binary *linking-decision* tree T_α whose nodes express linking decisions, blink(x_1,x_2), where the arguments x_1 and x_2 can be:

(a) a variable U_i,

(b) a node which is a proper descendant of some v_j node at the onset of the round, specified as being in a designated position (at the onset of the round) of the subtree rooted at the value of a designated variable, U_i (e.g. the node which was the second child of the third child of U_2 at the onset of the round),

(c) a node which is in the heap at the onset of the round, but not in the subtree of any v_i node, specified by its position in the heap at the onset of the round,

(d) a node specified as being the j^{th} node inserted since the onset of the round, for some $j \geq 1$.

Clearly the linking-decision root nodes of the various T_α are identical.

Lemma 4 *Suppose α_1 and α_2 are two assignments and that π_1 and π_2 are paths in the respective trees T_{α_1} and T_{α_2}. Suppose these paths are identical (the respective sequences of blinks and outcomes are identical) through their first k nodes, for some $k \geq 1$. On the basis of these k linkings we can infer the particular subset of the variables U_i whose values v_j have been deleted from the heap. Now suppose under the respective assignments α_1 and α_2 these deleted variables have been identically assigned. Then both paths π_1 and π_2 have identical $(k+1)$st nodes, or both terminate.*

Proof: This follows by induction on k using the following *fact*. Consider a linking-decision tree modeling a single deletemin operation for a root having a fixed number of children. The positions of the heap nodes accessed as arguments of a particular blink are a function only of the location of the blink in the linking-decision tree. $\square$

Remark. The *fact* mentioned in the above proof does not hold for the Fibonacci heap algorithm, and constitutes the essential reason why pairing heaps do not perform as efficiently as Fibonacci heaps.

Our goal is to meld the various trees T_α, for α belonging to an appropriate subset A of the possible assignments, into a single *ternary* linking-decision tree that simultaneously models the linkings that take place relative to the assignments $\alpha \in A$, assuming that our adversary guides the outcomes of these linkings. The expression tlink(U_i,x) (or tlink(x,U_i)), where x is of the form (b), (c) or (d) above (in other words, x is not a U_i variable) represents a ternary decision for the linking of U_i and x: the result is link(x,U_i), or link(U_i,x) with indication that this is an efficient linking for U_i, or link(U_i,x) with indication that this is *not* an efficient linking for U_i.

Given T_α, let $\bar{T}_\alpha$ denote the result of transforming T_α as follows. First, let π_α be the execution path in T_α determined by our adversary. Proceeding from the root of T_α, for each blink(x,U_i) (respectively blink(U_i,x)) on π_α, where x is not a U_j variable for some j, we replace the blink with the corresponding tlink; connecting the subtree of T_α corresponding to the outcome, link(U_i,x), to both of the branches of the tlink representing outcome link(U_i,x). The path π_α proceeds through the applicable branch of this tlink. The transformation of T_α into $\bar{T}_\alpha$, therefore, is accomplished by changing to ternary nodes certain nodes on the path π_α.

Lemma 5 *Assume that the minimum gap between the respective ranks of the nodes* $v_1, \cdots, v_p$ *is* g *with* $g > d$*, and that the ranks of these nodes are* $< L$*. Let* A *be the set of all assignments* α *such that in the course of the execution* π_α*, no* v_i *node has its rank increase by as much as* $g-d$ *before becoming efficient, and each* v_i *eventually has an efficient event. Let* α_1 *and* α_2 *be two assignments in* A *and let* π_{α_1} *and* π_{α_2} *be the respective execution paths in the linking-decision trees,* $\bar{T}_{\alpha_1}$ *and* $\bar{T}_{\alpha_2}$*. Suppose that the first* k *nodes and the first* $k-1$ *decisions of* π_{α_1} *and* π_{α_2} *are identical for some* $k \geq 1$*. Then*

(a) *Let* Γ *consist of the variables* U_i*, whose assigned values* v_j *have had an efficient event prior to the moment of the* k^{th} *linking. With respect to both assignments* α_1 *and* α_2 *the set* Γ *is the same and the variables in* Γ *are identically assigned. Moreover, for any operand, other than a* U_i *variable, appearing in a linking in this common portion of the paths* π_{α_1} *and* π_{α_2}*, the rank of the designated node at the relevant moment is independent of the particular assignment,* α_1 *or* α_2*, which is in effect.*

(b) *Suppose neither of the operands of the* k^{th} *node are* U_i *variables whose assigned values are pre-efficient at this point of the computation. (The tlink outcomes on the path leading to the* k^{th} *node determine whether the value of some* U_i *variable is pre-efficient.) Then the respective outcomes for the* k^{th} *node on the paths* π_{α_1} *and* π_{α_2} *are identical.*

Proof: First we establish (a). We observe that the variables U_i whose values v_j have been deleted from the heap are uniquely determined by the linkings that have taken place along the common portion of the paths π_{α_1} and π_{α_2}. For any other variable U_i in Γ whose value v_j has had an efficient event subsequent to the onset of the round, it is the case that v_j has had an efficient linking. The rank separation and gain constraint preclude the possibility that this efficient event for the value of U_i occurs in a linking of the form blink(U_i,U_h). Thus, the outcome of some linking-decision node of the tlink form, located prior to the k^{th} node on this common portion of the paths π_{α_1} and π_{α_2}, indicates that the value of U_i has an efficient linking. This establishes that Γ is the same for both assignments α_1 and α_2. To show that the respective assignments of the variables in Γ are the same, we argue by induction on k. We claim that only one v_i can have its next efficient event take place as a deletion, namely, the one with largest rank at the onset of the round. To be deleted in a deletemin operation, a node must first become the root of the heap, which by lemma 1 has rank $\geq L$. At most one v_i can gain sufficient rank to become a root while remaining pre-efficient due to the assumed separation of ranks and constraint on the growth of rank. In any other instance in which the value of some U_i becomes efficient prior to the linking of the k^{th} node, let link(U_i,x) be the efficient linking which is the outcome of the tlink in the corresponding linking-decision node. Since this linking takes place prior to the k^{th} node, our induction hypothesis implies that the rank of x at this point is independent of the particular assignment, α_1 or α_2, which is in effect. Because the rank of U_i is within d of the rank of x since link(U_i,x) is an efficient linking, the assumed separation of ranks and the assumed constraint on the growth of the rank of U_i's assigned value serve to fix the assigned value of U_i at the onset of the round, independently of the assignment α_1 or α_2 which is in effect. This establishes the uniqueness of the assignments of the variables in Γ.

Let x be an operand, other than a U_i variable, appearing in a linking in this common portion of the paths π_{α_1} and π_{α_2}. We proceed to establish that the rank of x is independent of the particular assignment, α_1 or α_2, which is in effect. By induction we may assume that x appears in the k^{th} node. We have two cases: (i) x does not belong to the subtree of any v_i at the onset of the round, and (ii) x is a proper descendant of some v_i at the onset of the round.

Suppose case (i) applies. Based on the location of x in this common portion of the paths π_{α_1} and π_{α_2}, we can deduce the structure of the subtree of x at this point in the computation in terms of various U_i variables that are linked in and various other entities that can appear as operands in a linking-decision tree. We note that the rank of x is uniquely determined by the indicated status (e-link or not e-link) of various links to its children and the ranks of the child nodes of the remaining links to its children. If a child of x is given by a U_i variable, then the status of the link can be deduced from the computation path (particularly whether or not the linking of U_i to x was indicated as being efficient). Now if a child y of x is not given as a U_i variable, then either it is one of the original children of x at the onset of the round, in which case its rank and is fixed, or y appeared as an operand when it was linked to x at an earlier point on the computation path. In this latter case, the rank of y is independent of the particular assignment, α_1 or α_2, which is in effect, by the induction hypothesis. We conclude that the rank of x is fixed, independently of the of the particlular assignment, α_1 or α_2, which is in effect.

Suppose case (ii) applies. In order for x to be participating in a linking it must be the case that the v_h ancestor of x had previously been deleted. Hence the variable U_i, which appears in the specification of x, has an assigned value which is independent of the particular assignment, α_1 or α_2, which is in effect. (This is a consequence of the sameness of the assignments to the variables in Γ, which we have already established.) Since x is specified by its position relative to its v_h ancestor, this serves to fix the identity of x. We now argue as in case (i) to conclude that the rank of x is fixed, independently of the particular assignment, α_1 or α_2, which is in effect.

To establish (b), we claim first that the rank of the value of any operand, other than a pre-efficient value of some U_i variable, which appears along the common portion of the paths π_{α_1} and π_{α_2}, is independent of the particular assignment, α_1 or α_2, which is in effect. This follows from part (a) of our lemma if the operand is not a U_i variable. Now consider an operand which is a U_i variable whose value has previously had an efficient event, and let v_h be the value assigned to U_i. (Part (a) implies that v_h is fixed.) We argue by induction on k that the rank of v_h is fixed at this point in the computation. The path leading to the linking-decision node containing the operand enables us to deduce the structure of the subtree of v_h. As in part (a), any child of v_h which is not the value of a U_j variable has fixed rank independently of the particular assignment, α_1 or α_2, which is in effect. If the child is designated as the value of some U_j variable, consider the prior node, blink(U_j, U_i), which causes the linking of this child. If the value of U_j is pre-efficient at that moment, which is discernable from the computation path, then the link is not an e-link as observed earlier. If the value of U_j has had an efficient event prior to that moment, then by the induction hypothesis, its rank is fixed at the moment of its linking to U_i. Thus, as in part (a) sufficient information exists about the links leading to the children of v_h to conclude that its rank is fixed, independently of the particular assignment, α_1 or α_2, which is in effect. This establishes our claim.

Now consider the k^{th} node of our common paths π_{α_1} and π_{α_2}. Our claim implies that the ranks of its operands are independent of the particular assignment, α_1 or α_2, which is in effect. But the outcome of the linking decision is dictated strictly by the ranks of these operands since our adversary is defined this way, and the indication whether or not a linking is efficient is dictated by the ranks of the operands. Thus the outcome of the linking is independent of the particular assignment, α_1 or α_2, which is in effect. This concludes the proof of the lemma. □

Let A be a set of assignments as described in lemma 5. We now proceed to construct a ternary linking-decision tree τ that simultaneously models the linkings that take place relative to the assignments $\alpha \in A$, assuming that our adversary guides the outcomes of the linkings. The tree τ will have the following properties.

(i) For each node ν of τ the path from the root of τ to ν, which for some k consists of k linking nodes and the outcomes of the first $k-1$ nodes, coincides with an initial portion of the path π_α (in $\bar{T}_\alpha$) for at least two assignments α in A.

(ii) For any assignment $\alpha \in A$, some initial portion of π_α (linking-decision nodes and their outcomes) coincides with a path in τ ending at a leaf.

(iii) A node of τ can have two or more children only if it involves the linking of some U_i variable whose value is pre-efficient (as determined by the path ending at that node).

(iv) The assignments in A are associated (as in property (ii)) with distinct paths in τ, so that τ has $|A|$ distinct leaves.

The root of τ is the common root of the $\bar{T}_\alpha$ trees. We proceed to build τ, replacing some leaf with a node at each step. Our inductive hypothesis is that properties (i) and (ii) hold after each step. For each leaf ℓ of τ, as constructed thus far, let A_ℓ denote the subset of $\alpha \in A$ associated with ℓ as in property (ii). Suppose $|A_\ell| \geq 2$. Lemma 4 and part (a) of lemma 5 imply that either (a) each π_α for $\alpha \in A_\ell$ simultaneously terminates at the parent of ℓ, or (b) these paths have with identical subsequent nodes. Our assumption that each v_i eventually has an efficient event, coupled with part (a) of lemma 5 implies that (a) is not possible. (Otherwise, the assignments in A_ℓ would be identical.) Thus, (b) holds. Let η be the common subsequent node on these paths. We then replace ℓ with η. Properties (i) and (ii) clearly remain in force. Property (iv) eventually holds; otherwise τ would grow indefinitely, contrary to the fact that the pairing heap algorithms terminate.

Now consider any node ν in τ that involves the linking of two heap nodes, neither of which is given as the pre-efficient value of some U_i variable. Lemma 5, part (b) implies that ν has only one child since only one outcome of the linking associated with ν can be realized. Thus, property (iii) holds.

Property (iii) asserts that our tree τ has branch points (nodes with two or more children) only at nodes that involve the linking of some U_i variable whose value is pre-efficient. The number of branch points in a given path ρ of τ therefore constitutes a lower bound for the total number of linkings that involve at least one pre-efficient v_i, for the assignment corresponding to ρ. The following lemma summarizes our construction.

Lemma 6 *Assume that restrictions R apply and that the minimum gap between the respective ranks of the nodes $v_1, \cdots, v_p$ is g with $g > d$, and that the ranks of these nodes are $< L$. Let A be the set of all assignments α such that in the course of the execution π_α, no v_i node has its rank increase by as much as $g - d$ before becoming efficient, and such that each v_i eventually has an efficient event. Then there exists a ternary linking-decision tree τ which models initial portions of the executions π_α for the assignments α in A. Our tree τ has distinct leaves corresponding to the assignments in A. Finally, for a given assignment α, the number branch points on the path identified with α gives a lower bound for the total number of linkings that involve at least one pre-efficient v_i.*

Lemma 7 *Assume that restrictions R apply and that the nodes $v_1, \cdots, v_p$ satisfy the conditions of lemma 6, where g has been chosen so that $g - d \geq p^{4/3}$. Assume that we stipulate an obligation of $\lceil p^{4/3} \rceil$ for each v_i that does not have a subsequent efficient event, remaining in the heap at the end of the O-sequence corresponding to a given assignment. Then upon averaging over the $p!$ possible assignments, the total renewed obligation of these nodes at the onset of the round is $\Omega(p \log_3 p)$.*

Proof: An assignment α of the v_j's to the U_i's is said to have *jumps* provided that at least one of the v_j's has a gain of at least $p^{4/3}$ in rank prior to becoming efficient. An assignment that has jumps obligates at least $p^{4/3}$ linkings for at least one of the v_i, since a single linking can increase the rank of a node by only one unit. Let A consist of the assignments for which no v_i has an actual or stipulated obligation as large as $p^{4/3}$. Without loss of generality we may assume that at least one-half of the $p!$ assignments belong to A. For each assignment in A, each v_i has a subsequent efficient event; an exceptional v_i would have a stipulated obligation of $p^{4/3}$. No assignment in A has jumps. Therefore A satisfies the hypothesis in lemma 6 since $g - d \geq p^{4/3}$. We apply lemma 6 to the set A, obtaining the ternary tree τ which models the executions π_α for the assignments in A. The average number of branch points on the paths corresponding to the assignments in A bounds the total obligation of the p nodes averaged over the assignments in A. Applying Kraft's inequality to bound this average, we conclude that the total obligation of the p nodes averaged over the assignments in A, is bounded below by $\log_3 |A| = \Omega(p \log_3 p)$. The conclusion of the lemma follows immediately. □

Summing

Lemma 8 *Assume that the decrease key operations in a given round select q nodes that have e-links to the root and ranks $< L$, and $q^{1/4} \geq 4(d+1)$. Assume that we stipulate an obligation of $\lceil q^{1/3} \rceil$ for each of these selected nodes that do not have a subsequent efficient event, remaining in the heap at the end of the O-sequence corresponding to a given continuation. Then this round contributes $\Omega(q \log_3 q)$ linkings (expected number) in obligations specified in the renewed schedules of these q items at the onset of the round.*

Proof: Let $p = \lfloor q^{1/4} \rfloor$. We begin by constructing disjoint subsets of our q nodes, with each subset having size p, such that the union of these subsets consists of $\geq q/2$ nodes, and such that the ranks of the nodes in each subset are separated by gaps of at least $2p^2$. These sets are constructed inductively as follows. Suppose that $\geq 2(d+1)p^3$ nodes among the initial q nodes remain, not having been placed in the subsets thus far constructed. By lemma 3 there are at most $d+1$ nodes of any given rank among our q nodes. Thus the ranks of our remaining nodes contain $\geq 2p^3$ distinct values, and it follows that we can select a subset of p nodes from these remaining nodes whose ranks are separated by gaps of at least $2p^2$. When this process finally stops, it is because there are fewer than $2(d+1)p^3$ remaining nodes, and our assumption implies that this quantity is $\leq q/2$. Let $g \geq 2p^2$ denote our guaranteed gap between ranks. Then $g - d \geq p^2$, and it follows that each of our selected sets of p nodes satisfies the hypothesis of lemma 7. Let S_j, $j \geq 1$, denote the subsets of p nodes that we have constructed. If we now fix j and condition upon an arbitrary instantiation I of the restrictions R defined at the beginning of our discussion of the ternary linking-decision tree, then upon averaging over the assignments of the nodes in S_j, we conclude from lemma 7 that these nodes incur as new obligations $\Omega(|S_j| \log_3 |S_j|) = \Omega(|S_j| \log_3 q)$ linkings, provided we stipulate an obligation of $\lceil |S_j|^{4/3} \rceil \leq \lceil q^{1/3} \rceil$ to each node of S_j that fails to have a subsequent efficient event, remaining in the heap at the end of O-sequence.

(Actually, only those instantiations compatible with the selection of our q nodes for decrease operations are of concern here.) Thus the overall expected contribution of the obligations of the nodes in S_j to the totol expected obligation of the q nodes is $\Omega(|S_j|\log_3 q)$. Summing over j we obtain our $\Omega(q\log_3 q)$ bound. This completes the proof. $\square$

Theorem The expected cost of executing an O-sequence of length $m \geq 2n$ is bounded by

$$\Omega(m(\log_3\log_3 n)^{1/2}) - O(n(\log_3 n)^{1/3}),$$

where m is the length of the sequence, and n is the number of nodes in the heap at the onset of each round.

Proof: If at the onset of a given round of an O-sequence the root of the heap has more than $C \cdot L$ children, then the execution of the deletemin at the conclusion of the round requires $C \cdot L$ linkings. If the root has $\leq C \cdot L$ children, then by lemma 2, the expected number $\bar{q}$ of nodes selected for decrease key operations that have e-links to the root and ranks $< L$ satisfies

$$\bar{q} \geq L/(2C), \tag{1}$$

provided $C \leq (3^d+1)/2$. Let p_q represent the probalitiy that the actual number of nodes selected for decrease operations that have e-links to the root and ranks $< L$ is q. Applying Jensen's inequality and lemma 8, taking into consideration the requirement in lemma 8 that the actual number q of selected children that have e-links to the root and rank $< L$ satisfy $q^{1/4} \geq 4(d+1)$, we conclude that the expected total of new obligations incurred by the round is bounded by

$$\Omega\Big(\sum_q p_q q\log_3 q - \sum_{q^{1/4}<4(d+1)} p_q q\log_3 q\Big) = \Omega(\bar{q}\log_3\bar{q} - \bar{q}\log_3(4^4(d+1)^4)). \tag{2}$$

Applying the inequality (1), choosing $C = \sqrt{\log_3 L}$ and $d = \lceil\log_3(2C)\rceil$ (the latter chosen so that the hypothesis of lemma 2 is satisfied), the right hand side of (2) is bounded by

$$\Omega(L\sqrt{\log_3 L}) = \Omega(C \cdot L).$$

Thus, if we let $\bar{\ell}_i$ denote the expected total amount of obligation incurred by the i^{th} round of an O-sequence and $\bar{c}_i$ denote the expected number of linkings that take place at the end of the i^{th} round (the number of children of the root), then we have established that

$$\bar{\ell}_i + \bar{c}_i = \Omega(L\sqrt{\log_3 L}). \tag{3}$$

Now for a particular O-sequence σ, let ℓ_i denote the total amount of obligation incurred by the i^{th} round and let c_i denote the number of linkings taking place in the i^{th} round. Summing over the rounds, we know that

$$\sum_i c_i \geq \Big(\sum_i \ell_i - u_\sigma\Big)/2,$$

where u_σ is the total amount of residual (unsatisfied) obligation residing in the heap at the conclusion of the O-sequence σ. We conclude that

$$3\sum_i c_i \geq \sum_i \ell_i + \sum_i c_i - u_\sigma . \quad (4)$$

Because no node remaining in the heap at the conclusion of an O-sequence has a stipulated obligation from lemma 8 exceeding $L^{1/3}$, we have $u_\sigma \leq nL^{1/3}$. Substituting into (4) and applying (3), we conclude that that the total expected cost of executing an O-sequence with r rounds is given by

$$\Omega(r \cdot L\sqrt{\log_3 L}) - O(nL^{1/3}).$$

Since a round consists of $L + 2$ operations, we conclude that the total expected cost of executing an O-sequence of length $m \geq 2n$ is bounded by

$$= \Omega(m(\log_3 \log_3 n)^{1/2}) - O(n(\log_3 n)^{1/3}),$$

completing the proof. □

Corollary It is not the case that pairing heaps perform insertion and deletemin operations in $O(\log_3 n)$ amortized time, and decrease key operations in constant amortized time when n items are in the heap.

4 Concluding Remark

The proof technique used in this paper applies to the variations on pairing heaps that have been suggested. Constant amortized time decrease key operations seem to present a challenge for self-adjusting implementations of heaps.

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.

[2] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization problems* JACM 34,3 (1987), pp. 596–615.

[3] A. M. Liao, *Three priority queue applications revisited*, Algorithmica 7,4 (1992), pp. 415–427.

[4] R. Sedgewick, **Algorithms**, Addison-Wesley, 1989.

[5] J. T. Stasko and J. S. Vitter, *Pairing heaps: experiments and analysis*, CACM 30,3 (1987), pp. 234–249.

[6] M. A. Weiss, **Data Structures and Algorithm Analysis in C**, Addison-Wesley, 1997.