1. INTRODUCTION

Let $U$ be a universe of $n$ elements. Suppose $U$ is partitioned into a collection of (named) singleton sets and suppose we want to be able to perform the following operations:

- Union($A$, $B$). Join the sets $A$ and $B$ (destroying sets $A$ and $B$), and relate a set name to the resulting set;
- Find($x$). Return the name of the set in which element $x$ is contained.

The occurring set names must satisfy the condition that, at every moment, the names of the existing sets are distinct. The problem of efficiently implementing Union–Find programs is widely known as the “disjoint set union problem” or the “UnionFind problem.”

Several algorithms for the Union–Find problem have been developed. In 1975 (cf. [14]) Tarjan considered the well-known set union algorithm that uses path compression. He proved that the worst-case time bound for this algorithm is $O(m \cdot \alpha(m, n))$ for $n$ unions and $m \geq n$ finds, where $\alpha$ is the inverse Ackermann function. The algorithm can run on a pointer machine (i.e., a machine model of which the memory consists of records, each containing a bounded number of pointers [8, 9, 13, 15]). There are several other Union–Find algorithms that run on pointer machines in the above time bound and that use a form of path compaction [16]. In [10] a new algorithm without path compaction (with a similar approach as in [5]) is presented that runs on a pointer machine and that has a worst-case time bound of $O(f(n))$ for the $f$th Find, within the bound of $O(n + m \cdot \alpha(m, n))$ time for $m$ finds on $n$ elements as a whole.

In 1979 (cf. [15]) Tarjan proved a lower bound on the time complexity of Union–Find programs on a pointer machine that satisfy the separation condition (which is defined below): such a program of $n - 1$ unions and $m$ finds takes at least $O(m \cdot \alpha(m, n))$ time, if $m \geq n$. In [1, 16] the bound was extended to $O(n + m \cdot \alpha(m, n))$ time for all $n$ and $m$. The proof of the bound relies heavily on the separation condition (cf. [15]):

At any time during the computation, the contents of the memory can be partitioned into collections of records such that each collection corresponds to a currently existing set, and no record in one collection contains a pointer to a record in another collection.

As shown in [12], the separation condition can imply a loss of efficiency (cf., e.g., Table 1). Hence, the lower bound of [15] is not general enough for pointer machines. Moreover, not all known Union–Find algorithms that run on a pointer machine satisfy the separation condition: the algorithm in [10] does not satisfy the separation condition since a list of all records with set names needs to be used. However, since the list is not used for finds, the model in [15] can be liberalized such that the algorithm implies a modified algorithm with the same time bound that does satisfy the conditions.

In this paper we prove a $O(n + m \cdot \alpha(m, n))$ lower bound for the Union–Find problem on a general pointer machine (without the separation condition). A consequence of the lower bound is that the Union–Find algorithms given in [10, 14, 16] are optimal for pointer machines.

The related problem is the Split–Find problem. Let $U$ be a totally ordered universe of $n$ elements. Suppose $U$ is partitioned into a collection of (named) sets and suppose we want to be able to perform the following operations:

* An extended abstract of this paper appeared in the “Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC), May 1990, Baltimore, Maryland,” where it received the Best Student Paper Award.

† Full name: Johannes A. La Poutré.
• Split(x). Split the set in which x is contained into two sets, one set consisting of all elements in the set \( x \), and the other set consisting of the remainder; relate a set name to each of these new sets;
• Find(x). Return the name of the set in which element x is contained.

Again, the occurring set names must satisfy the condition that, at every moment, the names of the existing sets are distinct.

In [7] an algorithm for the Split–Find problem was presented that runs in \( O(n + m \cdot \log^* n) \) time on a pointer machine (and that satisfies the separation condition). In [5, 11] algorithms for the Split–Find problem are presented that run in \( O(n + m \cdot x(m, n)) \) time on a pointer machine. Until now no lower bound was found for the Split–Find problem on a pointer machine.

We prove a \( \Omega(n + m(m, n)) \) lower bound for the Split–Find problem on general pointer machines, too. A consequence of the lower bound is that the Split–Find algorithms given in [5, 11] are optimal for pointer machines.

Our proofs use inductive structures that are related to the inductive structures used in [5, 10, 11]. The lower bounds are proved for all possible sequences of Unions (c.q. Splits) that are in some class of “balanced” sequences of Unions (c.q. Splits) and that may be known in advance; each such sequence can be intermixed with appropriate FINDs to yield the lower bound. Some consequences are that the special cases of the Union–Find problem that can be solved in linear time on a RAM (cf. [6]) (viz., where the structure of the (arbitrary) Union sequence is known in advance) do not have a linear solution on a pointer machine and that, (arbitrary) Union sequence is known in advance) do not have a linear solution on a pointer machine and that, although the Split–Find problem can be solved in linear time on a RAM (cf. [6]), this is not possible on a pointer machine.

Recently, in [4] a lower bound was proved for the Union–Find problem on the Cell Probe Machine with word size \( \log n \), where \( n \) is the size of the universe. Our result does not use any restrictions on the word size (and thus any amount of additional information can be used freely), but it is only based on properties of addressing by means of pointers instead. Some previous other lower bounds for the Union–Find and the Split–Find problem on pointer machines were given for the worst-case time of the Union–Find problem on a pointer machine with the separation condition [2] and the worst-case time of the Split–Find problem [12]. Table I gives an overview of the existing and new results for lower bounds on pointer machines. As remarked in [12], it appears that the separation condition can imply a loss in efficiency (like, e.g., in the worst-case Split–Find bounds).

As remarked by Tarjan in [15], for each individual Union–Find problem on \( n \) elements there exists a dedicated pointer machine that executes the operations in it in linear time. (Viz., take a pointer machine with \( n \) pointers per node and link each element to a central node and link the central node to each set name.) Therefore, it is not possible to have a non-trivial general lower bound for all pointer machines with a varying number of pointers per node. (Note that this observation holds for all related problems too, including worst-case problems.) Tarjan conjectured that for an individual pointer machine the \( x \)-bound should hold. In this paper we prove that this bound holds indeed, and, moreover, we show that there is a general constant \( d \) that holds for all pointer machines, such that a lower bound of \( d(n + m \cdot x(m, n)) \) steps holds for all \( m \) and \( n \). This implies that there is no "asymptotic speed up" for the Union–Find problem if we increase the maximal number of pointers per node in a pointer machine. Note that this is the strongest asymptotic result that is possible. The same observations can be made w.r.t. the Split–Find problem.

The paper is organized as follows. In Section 2 pointer machines and the Ackermann function are considered. In Section 3 we define some notions w.r.t. Unions and we introduce machines for which we prove lower bounds in Section 4. In Section 5 the actual lower bounds for the Union–Find problem are proved. In Section 6 the lower bounds for the Split–Find problem are proved.

2. PRELIMINARIES

2.1. Pointer Machine Model

The computational model we use is a liberal version of the pointer machine as described in [15]. (Also cf. [8, 9, 13].) A pointer machine consists of a collection of nodes. A pointer is the specification of some node (namely, of the node pointed to, and thus not being nil). Each node contains \( c \) fields that each may contain either one pointer or the value nil (\( c \geq 1 \)). The instructions that a pointer machine can execute are of the following types:

- the creation of a new node with nil in all its fields,
- a change of the contents of a field of a node.

We call a pointer machine with \( c \) fields per node a \( c \)-pointer machine. A program is a sequence of instructions to be executed by a pointer machine. (The instructions given above are more liberal than those in [15] since we do not specify the way of addressing in pointer machines yet; we
will condense this way of addressing in the definition of the cost of the operation Find. Furthermore, we do not consider an output instruction explicitly.)

A pointer machine can be regarded as a dynamic directed graph when a pointer to node \( y \) in a field of some node \( x \) is represented by an edge \((x, y)\). A path from node \( x \) to node \( y \) is a sequence of nodes such that each node contains a pointer to its successor in the sequence and the first and last node of the sequence are \( x \) and \( y \), respectively. The length of a path is the number of nodes in it, not counting its first node. The distance from \( x \) to \( y \) is the minimum length of any path from \( x \) to \( y \).

The \textit{Union–Find problem} on a pointer machine is, as usual, given as follows (also see [15] or [12, 14, 16]). Let \( U \) be a collection of nodes, called elements. Let \( U \) be partitioned into a collection of sets, and let to each set a (possibly new) unique node be related, called “set name.” This partition is called the initial partition. (For the regular Union–Find problem the sets in the partition are singleton sets; however, for convenience in our analysis, we allow other partitions too.) The problem is to carry out a sequence of the following operations:

- \textit{Union}(\( A, B \)). Join the sets \( A \) and \( B \) (destroying the old sets \( A \) and \( B \)) and relate a set name to the resulting set.
- \textit{Find}(\( x \)). Return the name of the current set in which element \( x \) is contained.

The operations are carried out semi-on-line: i.e., each operation must be completed before the next operation is known, while the subsequence of Unions may be known in advance. Moreover, occurring set names must always satisfy the condition that, at every moment, the names of the existing sets are distinct. (Henceforth, we will not state the latter condition explicitly any more, but just consider this to be part of the definition of set name.)

An execution of a sequence of Union and Find operations on a pointer machine consists of a (so-called initial) contents of the pointer machine, together with a sequence of programs that carries out the Union–Find problem according to the following rules:

1. Initially, before the first operation is carried out, the contents of the pointer machine, called the initial contents, \( \text{reflects} \) the initial partition of the universe: i.e., for each element there exists a path to the (unique) name of the set in which it is contained.

2. Each Union is carried out by executing a Union program, which halts having modified the contents of the pointer machine to reflect the Union (where some node is indicated as the name of the resulting set) and, hence, to reflect the new partition of the universe.

3. Each Find is carried out by executing a Find program, which halts having identified the name of the set containing the considered element while the pointer machine still reflects the (unchanged) partition of the universe.

4. For each Union or Find operation in the sequence, the corresponding program is not executed until the program of its predecessor operation has halted.

The \textit{cost} of an execution of a sequence of Union and Find operations is the cost of the Union and Find operations, which are defined as follows:

- the cost of a Union is the number of pointer addings: i.e., changes in fields that change the contents of a field (whatever the contents were) into some pointer (hence, not \textit{nil}).
- the cost of \textit{FIND}(\( x \)) is the length of the shortest path from \( x \) to its set name at the start of the Find, together with the number of pointer addings performed during the Find.

Then the number of (pointer machine) \textit{steps} performed during the execution of a Union–Find problem certainly is at least the cost of that execution, with a minimum of one step per operation. (We will use the notion of steps only in some final theorems.)

We state some observations. Note that in our complexity measure (viz., cost and number of steps) we do not account for any change of the contents of a field to \textit{nil}. This is for convenience, and also it expresses the role of “real” pointer values (as opposed to field value \textit{nil}). Also, note that the mere fulfillment of requirement 2 requires only \( O(1) \) cost per operation (viz., by \( O(1) \) pointer addings), and that additional modifications could be done later, in other operations (as happens in, e.g., path compaction algorithms). Furthermore, we note that the name of the set resulting from a Union may be specified by a user, and that, if wanted, this set name may be returned by the Union operation; this does not influence our considerations, since we do not account for these costs. Finally, note that we consider a slightly relaxed version of the lower bound model for the Union–Find problem on pointer machines in comparison with, e.g., [15, 12], since we do not account for all possible machine operations in the complexity measure. (And, of course, our lower bounds thus directly hold in these other descriptions [15, 12] as well.)

2.2. The Ackermann Function

The Ackermann function \( A \) is defined as follows. For \( i, x \geq 0 \) function \( A \) is given by

\[
A(0, x) = 2x \quad \text{for } x \geq 0 \\
A(i, 0) = 1 \quad \text{for } i \geq 1 \quad (1)
\]

\[
A(i, x) = A(i-1, A(i, x-1)) \quad \text{for } i \geq 1, x \geq 1.
\]

The row inverse \( a \) of \( A \) and the functional inverse \( \alpha \) of \( A \) are defined by

\[
a(i, n) = \min \{ x \geq 0 | A(i, x) \geq n \} \quad (i \geq 0, n \geq 0) \quad (2)
\]

\[
\alpha(m, n) = \min \{ i \geq 1 | A(i, 4[m/n]) \geq n \} \quad (m \geq 0, n \geq 1). \quad (3)
\]
Here we take \([0] = 1\). (Note that \(x(0, n) = x(n, n)\).) The above two definitions correspond to those given in [5, 10, 11]. It is easily shown that the differences with the definitions given in [14–16] are bounded by some additive constants (except for \(a(0, n)\) and \(a(1, n)\)). We quote some results from [10].

It is easily seen that \(A(i, 1) = 2\), \(A(i, 2) = 4\), \(A(i, 3) = A(i, 4)\), and \(A(i + 1, 4) = A(i, A(i, 4))\) for \(i \geq 0\).

**Lemma 2.1.**

\[
A(i', x') \geq A(i, x) \quad (i' \geq i, x' \geq x)
\]

\[
a(i, n) \leq a(i', n') \quad (i' \geq i, n' \geq n)
\]

\[
x(m', n') \leq x(m, n) \quad (m' \geq m, n' \leq n).
\]

**Lemma 2.2.**

\[
a(i, A(i, x)) = x \quad (i \geq 0, x \geq 0)
\]

\[
a(i, A(i + 1, x)) = A(i + 1, x) \quad (i \geq 0, x \geq 0)
\]

\[
a(i, n) = a(i, A(i - 1, n)) + 1 \quad (i \geq 1, n \geq 2).
\]

**Proof.** By (1) we have \(a(i, A(i + 1, x + 1)) = a(i, A(i, A(i + 1, x))) = A(i + 1, x)\). Moreover, since \(n \geq 2\) implies \(a(n, 1) \geq 1\) and by (2), (1), and \(i \geq 1\) we find

\[
a(i, n) = \min\{ j \geq 1 \mid A(i, j) \geq n \}
\]

\[
= \min\{ j \geq 1 \mid A(i - 1, A(i, j - 1)) \geq n \}
\]

\[
= \min\{ j \geq 1 \mid A(i, j - 1) \geq a(i - 1, n) \}
\]

\[
= \min\{ j' \geq 0 \mid A(i, j') \geq a(i - 1, n) \} + 1
\]

\[
= a(i, A(i - 1, n)) + 1.
\]

**Lemma 2.3.** Let \(A^{(0)}(i, y) := y\) and \(A^{(i + 1)}(i, y) := A(i, A^{(i)}(i, y))\) for \(i, x, y \geq 0\). Then \(A(i, x) = A^{(i)}(i - 1, 1)\) for \(i \geq 1\), \(x \geq 0\). Let \(A^{(0)}(i, n) := n\) and \(A^{(i + 1)}(i, n) := a(i, A^{(i)}(i, n))\) for \(i, j \geq 0\), \(n \geq 1\). Then \(a(i, n) = \min\{ j \mid A^{(i)}(i - 1, n) = 1 \}\) for \(i \geq 1\).

By Lemma 2.3 it follows that for every \(i\), \(A(i + 1, x)\) is the result of \(x\) recurrent applications of function \(A(i, \cdot)\). Hence, we have

\[
A(0, x) = 2x
\]

\[
A(1, x) = 2^x
\]

\[
A(2, x) = 2^{2^x}
\]

\[
A(3, x) = 2^{2^{2^x}}
\]

Moreover, for \(i \geq 1\), \(x \geq 0\) the number \(A(i, x)\) is some power of two.

On the other hand, we have for \(n \geq 1\),

\[
a(0, n) = \lceil n/2 \rceil
\]

\[
a(1, n) = \lceil \log n \rceil = \min\{ j \mid \lceil j/2 \rceil = 1 \}
\]

\[
a(2, n) = \log^* n = \min\{ j \mid \lceil \log^*(j) \rceil = 1 \}
\]

\[
a(3, n) = \min\{ j \mid \log^{*(j)} n = 1 \}
\]

where as usual, the superscript \((j)\) denotes the function obtained by \(j\) consecutive applications.

By means of the row inverse of the Ackermann function we can express the functional inverse \(x\) as follows.

**Lemma 2.4.** \(x(m, n) = \min\{ i \mid a(i, n) \leq 4 \cdot \lfloor m/n \rfloor \} - 1\).

We state some lemmas that we will need in the sequel. The proofs can be skipped at first reading. The lemmas use

\[
n \geq 3 \Rightarrow i + 1 \Rightarrow a(i, n) = A(i + 1, a(i, n) - 2).
\]

This follows by using Lemma 2.2 that gives \(a(i + 1, a(i, n)) = a(i + 1, n - 1)\) and by using (2).

**Lemma 2.5.** For \(n\) and \(c\) such that \(x(n, n) > x(c, c) + 1\) the following holds for \(i\) with \(1 \leq i < x(n, n) - 3\):

\[
a(i, n) \geq 8 \cdot 12^i \cdot (c + 1)^{i - 1} \cdot (2a(i, n) + c + 1).
\]

**Proof.** Let \(n\) and \(c\) satisfy \(x(n, n) > x(c, c) + 2\). (Hence \(x(n, n) \geq 3\).)

**Claim 2.6.** \(c + 1 \leq a(i, n)\) for \(i\) with \(1 \leq i < x(n, n) - 2\).

**Proof.** By (3) we have \(A(x(n, n) - 1, 4) < n\) and \(A(x(c, c), 4) < c\). By using \(x(n, n) - 2 \geq x(c, c)\) it follows that

\[
n > A(x(n, n) - 1, 4) = A(x(n, n) - 2, A(x(n, n) - 2, 4))
\]

\[
\geq A(x(n, n) - 2, c).
\]

Hence by (2) we obtain \(c < a(x(n, n) - 2, n)\). By Lemma 2.1 it follows that \(c < a(i, n)\) for \(i \) with \(1 \leq i < x(n, n) - 2\). This proves the claim.

**Claim 2.7.** \(A(i + 1, x - 2) \geq 2 \cdot 12^{i+1} \cdot x^i\) for \(x \geq 6\) and \(i \geq 1\).

**Proof.** We prove the claim by induction to \(i\) and \(n\). First, for \(i = 1\) and \(x = 6\) we have \(A(2, 4) = A(1, A(1, 4)) = 2^{2^2} \geq 2 \cdot 12^1 \cdot 1.6\). For \(i > 1\) and \(x = 6\) we have by induction

\[
A(i + 1, 4) = A(i, A(i, 4)) \geq 2^{2 \cdot 12^{i+1} \cdot (i - 1)^{i - 1}} \geq 2 \cdot 12^{i+1} \cdot i.6^i.
\]
Finally, for \( i > 1 \) and \( x > 6 \) we have by induction

\[
A(i + 1, x - 2) = A(i, A(i + 1, x - 3)) \geq 2^{A(i + 1, x - 3)} \geq 2^{2.12^{i+1} \cdot i \cdot (x - 1)^{i'}} \geq 2.12^{i+1} \cdot i \cdot x^i.
\]

This proves the claim. 

Let \( i \) be such that \( 1 \leq i \leq n(n, n) - 3 \). Note that \( i + 2 \leq n(n, n) - 1 \) implies \( a(i + 2, n) \geq 5 \) and \( n \geq 3 \). By applying (4) for \( i + 1 \) we obtain

\[
a(i + 1, n) > A(i + 2, a(i + 2, n) - 2) \geq A(i + 2, 5 - 2) \geq A(3, 3) \geq 6.
\]

Hence, Eq. (4) and Claim 2.7 give that

\[
a(i + 1, n) > 2.12^{i+1} \cdot i \cdot (a(i + 1, n))^i \geq 8.12^{i+1} \cdot i \cdot (a(i + 1, n))^i \geq 2.12^{i+1} \cdot i \cdot x^i.
\]

By Claim 2.6 the inequality of Lemma 2.5 follows. 

LEMMA 2.8. Let \( n \geq 0 \), \( 1 \leq i \leq n(n, n) - 2 \). Then

\[
a(i + 1, n) < a(i, n)/i.
\]

Proof. Since \( i + 2 \leq n(n, n) \) we have \( a(i + 1, n) \geq 5 \) and \( n \geq 3 \). Claim 2.7 gives that \( A(i + 1, x - 2) \geq i \cdot x \) for \( x \geq 6 \) and \( i \geq 1 \). Moreover, \( A(i + 1, 5 - 2) = A(i, 4) \geq i \cdot 5 \) by Claim 2.7 and by \( A(1, 4) = 16 \). Applying this in (4) yields the required result. 

3. TURN SEQUENCES AND GU(\( i, c, p \)) MACHINES

In this section and in the following section we only consider the Union operation and a related operator. Consider a universe \( V \). Let \( US \) be a sequence of Unions on \( V \) starting from partition \( P \) and resulting in partition \( P' \). We represent each Union by the pair \( (A, B) \) of the two sets \( A \) and \( B \) that are joined by it. Henceforth we use the thus obtained sequence \( ((A_k, B_k)) \) to denote the Union sequence \( US \). We call \( US \) to be complete if \( P \) consists of singleton sets and \( P' = \{V\} \).

Suppose universe \( V \) has \( 2^x \) elements (for some integer \( x \)). Let \( P \) be a partition of \( V \) into sets of size \( 2^y \) (for some integer \( a \)). A Union Turn or 0-Turn \( T \) with initial partition \( P \) is an unordered collection of pairs \( (A, B) \) of sets \( A, B \in P \) such that each set in partition \( P \) occurs exactly once in the collection of pairs. (The Union Turn actually denotes the joining of the paired sets.) Partition \( P' = \{A \cup B | (A, B) \in T\} \) is called the result partition of \( T \) (consisting of sets of size \( 2^{x+1} \)). A 0-Turn sequence \( TS = (T_i) \) is a sequence of 0-Turns \( T_i \), such that the result partition of any 0-Turn is the initial partition of the following 0-Turn (if any).

Now consider some subuniverse \( U \subseteq V \) and some \( \alpha, 0 \leq \alpha < 1 \) with \( |U| \geq (1 - \alpha) \cdot |V| \). Consider a 0-Turn \( T \) on \( V \). Then the restriction of \( T \) to \( U \) is given by

\[
T|_U = \{(A \cap U, B \cap U) | (A, B) \in T\}.
\]

We call \( T|_U \) an \( \alpha \)-Turn or a just a Turn. The initial partition of \( T|_U \) consists of all non-empty sets occurring in the Turn and the result partition is the collection \( \{A \cup B | (A, B) \in T|_U \land A \cup B \neq \emptyset\} \). We call the sets in such a partition of \( U \) to have \( \alpha \)-size \( 2^x \) if the sets in the corresponding partition of \( V \) have size \( 2^x \). (Note that the actual universe \( V \supseteq U \) does not need to be known explicitly: \( a \) follows directly and uniquely from the partition of \( U \), since by \( 0 \leq \alpha < 4 \) the partition consists of sets of size \( \leq 2^x \), of which at least one must have size \( > 2^{x-1} \).) Now consider a 0-Turn sequence \( TS = (T_i) \) on \( V \). Then the sequence \( TS|_U := (T_i|_U) \), is called an \( \alpha \)-Turn sequence on universe \( U \). The initial partition of the sequence is the initial partition of its first Turn and the result partition of the sequence is the result partition of its last Turn. Note that both the universe \( U \), the initial partitions and the final partition are completely determined by the \( \alpha \)-Turn sequence. A 0-Turn sequence is called to be complete if the initial partition consists of singleton sets and the result partition consists of one set.

The operation \( \alpha \)-Turn \( T \) is given by: for each pair \( (A, B) \in T \) \( (A \neq \emptyset \lor B \neq \emptyset) \), join the sets \( A \) and \( B \) (destroying the old sets \( A \) and \( B \) if both \( A \) and \( B \) are non-empty) and relate some set name to the resulting set \( A \cup B \). (Note that if, e.g., \( A \neq \emptyset = B \) then set \( A \) remains unchanged, but it may get a new name.)

We now consider the actual executions of sequences as described above. An execution of a Union sequence \( US \) is defined as an execution of a sequence of Union and Find operations (as defined in Subsection 2.1) consisting of the Union sequence \( US \) only, where the non-occurrence of the Find operations in the sequence may be known in advance, i.e., at the start of the sequence. (Hence, because of the semi-online condition, the entire Union sequence may be known in advance; see Subsection 2.1.) An execution of an \( \alpha \)-Turn sequence on a pointer machine consists of a (so-called initial) contents of the pointer machine, together with a sequence of executions of \( \alpha \)-Turn operations according to the following rules:

1. Initially, before the first operation is carried out, the contents of the pointer machine (called the initial contents) reflects the initial partition of the universe: i.e., to each nonempty set some (unique) set name is related and for each element there exists a path to the name of the set in which it is contained.

2. Each \( \alpha \)-Turn is carried out by executing a program, which halts, having modified the contents of the pointer
3. For each operation in the sequence, the corresponding program is not executed until the program of its predecessor operation has halted.

The above executions are called \( UF(i, c) \)-executions if the executions are performed on a \( c \)-pointer machine and if initially (i.e., when the pointer machine reflects the initial partition) and at the end of each operation (i.e., when the pointer machine reflects the partition resulting from the operation) each element has distance at most \( i \) to its set name.

Let \( TS \) be a 0-Turn sequence. Then a Union sequence obtained from \( TS \) by replacing each Turn by a sequence of its pairs is called an implementation of \( TS \). A Union sequence is called balanced if it is an implementation of a complete 0-Turn sequence. A Union sequence on a universe \( U \) of \( n \) elements is called sub-balanced if it is a complete Union sequence on \( U \) that consists of a balanced Union sequence on some subuniverse \( V \subseteq U \) with \( |V| \geq \eta n \) that is intermixed with additional Unions. Obviously, for any universe there exists a sub-balanced Union sequence on it.

**Lemma 3.1.** Let \( TS \) be a complete 0-Turn sequence. Let \( US \) be a Union sequence that is an implementation of \( TS \). Let \( E \) be a \( UF(i, c) \)-execution of \( US \). Then there exists a \( UF(i, c) \)-execution of \( TS \) with cost that is at most the cost of \( E \).

**Proof.** The \( UF(i, c) \)-execution \( E \) is a valid execution of \( TS \) if all instructions in \( E \) for the Unions corresponding to one Turn are executed consecutively as one program.

**Definition 3.2.** Let \( i \geq 1 \) and \( 1 \leq c \leq p \). A GU(i, c, p) machine \( G \) (generic union machine) is a pointer machine that is used for the execution of an \( i \)-Turn sequence and for which the following constraints and modifications hold:

1. At any moment the collection of nodes in \( G \) is partitioned into \( i+1 \) disjoint sets, called layers. The layers are numbered from 0 to \( i \). Every node remains in the same layer.
2. At any moment set names are in layer 0 and elements are in layer \( i \).
3. Nodes in layer 1 have \( p \) fields and all other nodes have \( c \) fields.
4. A field of a node in layer \( j (0 \leq j \leq i) \) contains either the value nil or a pointer to a node in layer \( j+1 \) (if \( j \geq i \)).

**Lemma 3.3.** Let \( TS \) be a 0-Turn sequence on a universe \( U \) of \( n \) elements (\( n \) is a power of two). Let \( E \) be a \( UF(i, c) \)-execution of \( TS \) and let \( C \) be the cost of \( E \). Then there exists an execution \( EE \) of \( TS \) on a GU(i, c + 1, c + 1) machine \( GG \) such that initially in \( GG \), when \( GG \) reflects the initial partition of \( TS \), there are at most \( 2(c+1)^{i+1}, n \) fields that contain a pointer and such that \( EE \) has cost that is at most \( 2(i)(c+1)^{i+1}, C \) if \( i \geq 2 \) and at most \( C \) if \( i = 1 \).

**Proof.** Let \( G \) be a \( c \)-pointer machine on which execution \( E \) is performed. Let \( c \) fields of a node be numbered from 1 to \( c \). We first derive an execution \( EE \) on a GU(i, c + 1, c + 1) machine \( GG \) from \( E \). Every node \( x \) in \( G \) has for each \( 0 \leq j \leq i \) a (fixed) representative node \( x_j \) in layer \( j \) of \( GG \) and each node in \( GG \) is a representative of one node in \( G \). Let the fields of a node in \( GG \) be numbered from 0 to \( c \). Then execution \( EE \) is obtained from \( E \) by maintaining the following relations:

- For each node \( x \) in \( G \) the representative \( x_i \) in \( GG \) with \( 1 \leq j \leq i \) contains a pointer to the representative \( x_{j-1} \) in its 0th field;
- If in \( G \) node \( x \) contains a pointer to node \( y \) in its \( j \)th field \( (1 \leq a \leq c) \), then in \( GG \) node \( x_j \) \((1 \leq j \leq i)\) contains a pointer to \( y_{j-1} \), in its \( a \)th field;
- All other fields in \( GG \) contain nil.

The elements in \( GG \) are the representatives \( e_i \) of the elements \( e \) in \( G \) (i.e., these nodes \( e_i \) are identified with each other). The set names in \( GG \) are the representatives \( x_0 \) of nodes \( x \) that occur as set names in \( G \). We describe how to obtain an execution \( EE \) on \( G \). Each node \( x \) in \( GG \) has at most one representative node \( x \) in \( GG \) and, conversely, each node in \( GG \) is the representative of exactly one node in \( GG \). Moreover, node \( x \) in \( GG \) is in the same layer as its original \( x \) in \( GG \). Then execution \( EE \) is obtained from \( EE \) by the following rules:

- The initial contents of \( GG \) consist of those nodes \( x \) for which node \( x \) in the initial contents of \( GG \) is reachable from some element in \( GG \) (i.e., there exists a path in \( GG \) from some element to \( x \)).
- At the end of each operation \( GG \) contains all nodes \( x \) that either existed in \( GG \) at the start of that operation or of which the (possibly just created) original \( x \) in \( GG \) is reachable from some element in \( GG \) at the end of that operation in \( EE \).
- Initially and at the end of each operation the contents of the fields satisfy: if in \( GG \) the \( a \)th field of node \( x \) contains a pointer to node \( y \), then in \( GG \) the \( a \)th field of node \( x \) (if present) contains a pointer to node \( y \) (if present) and it contains nil otherwise.

Note that a node in \( GG \) can only become reachable from some element in \( GG \) if some pointer adding occurs in a field in \( G \). Each field in \( G \) corresponds to at most \( i \) fields outside layer 0 in \( GG \). Each pointer added in such a field points to a node in a layer \( j \) with \( 0 \leq j \leq i \). Moreover, a node in layer \( j \) with \( 0 \leq j \leq i \) of \( GG \) has at most \( \sum_{j=0}^{i} (c+1)^{i-j} - 1 \leq 2(c+1)^{i+1}-1 \) nodes outside layer 0 of \( GG \) that are reachable from it. Therefore it follows that any
LOWER BOUNDS FOR THE UNION-FIND PROBLEM

In this section we will prove lower bounds for executions of \(\alpha\)-Turn sequences on GU\((i, c, p)\) machines. We will use these results in Section 5, viz., in deriving lower bounds for UF\((i, c)\)-executions of Union sequences and, subsequently, for the Union–Find problem itself.

**Lemma 4.1.** Let \(G\) be a GU\((1, c, p)\) machine. Let \(TS\) be an \(\alpha\)-Turn sequence for some \(\alpha, 0 \leq \alpha \leq \frac{1}{2}\), and let \(n\) be the number of elements. Suppose the initial partition consists of sets of size \(\leq 2^\alpha\) and the result partition consists of sets of size \(\geq 2^n\). Suppose \(q_1 - q_0 \geq 4p\). Let \(E\) be an execution of TS on \(G\). Then at least \(\frac{n}{2} \cdot (q_1 - q_0)\) pointer addings occur in \(E\).

**Proof.** Let \(U\) be a universe of elements of \(G\). By the definition of \(\alpha\)-Turn sequence, there exists a universe \(V \supseteq U\) and a 0-Turn sequence \(TSO\) on \(V\) such that \(TS = TSO|_V\). Let \(n \geq (1 - \alpha)|V|\). Let integer \(v\) be given by \(|V| = 2^v\). Hence, \(n \geq (1 - \alpha)2^v\).

We define a so-called matching sequence of an execution of TS or TSO as follows. Let \(TT\) denote TS or TSO. Let \(EE\) be an execution of \(TT\) on \(G\). First, for a Turn \(T\) in \(TT\) a matching sequence for \(T\) w.r.t. \(EE\) is a sequence that contains all the pairs \((e, s)\) of elements \(e\) and set names \(s\) such that \(e\) is the set name for \(s\) at the end of the program for \(T\) in \(EE\). A matching sequence of \(EE\) is a sequence of pairs obtained by replacing each Turn \(T\) in \(TT\) by a matching sequence for that Turn w.r.t. \(EE\).

Consider an execution \(EE\) of TSO on \(G\). Let \(M\) be a matching sequence of \(EE\). Then it obviously consists of \((q_1 - q_0)2^v\) pairs. For some node \(s\) that occurs as a set name in \(M\), consider the last time that \(s\) is the name of a set in \(M\). Let \(A\) be this set and let \(T_A\) be the 0-Turn that yields set \(A\). Suppose \(A\) has \(2^v\) elements. Then the matching sequence for \(T_A\) occurring in \(M\) contains \(2^v\) different pairs with \(s\) as set name. For all 0-Turns preceding \(T_A\) at most one set per 0-Turn has \(s\) as its set name. Therefore, at most \(1 + 2 + 2^2 + \cdots + 2^{v-1} = 2^v - 1\) pairs in \(M\) contain \(s\) as set name before the matching sequence for \(T_A\) occurs in \(M\). (These pairs may contain elements of \(A\).) Therefore at least half of the pairs in \(M\) that contain \(s\) as set name are distinct. Hence, the number of distinct pairs of elements and set names in \(M\) is at least \(\frac{1}{2} \cdot (q_1 - q_0)2^v\).

Hence, any matching sequence of an execution of TSO contains at least

\[
\frac{1}{2} \cdot (q_1 - q_0)2^v
\]

different pairs.

Consider the execution \(E\) of \(TS\). Let \(M\) be a matching sequence for \(E\). Note that \(E\) can be augmented to be an execution of TSO by performing at most \((q_1 - q_0)(2^\alpha - n)\) pointer addings in the \(2^\alpha - n\) elements of \(V\) during the Turns. Then \(M\) appropriately intermixed with \((q_1 - q_0)(2^\alpha - n)\) pairs for the elements in \(V\) \(\setminus U\) is a matching sequence of the resulting execution of TSO. By (5) this gives that there must be at least \(\frac{1}{2} \cdot (q_1 - q_0)(2^\alpha - 2^n)\) different pairs in \(M\).

Note that each pair \((x, s)\) in matching sequence \(M\) corresponds to a pointer to set name \(s\) in some field of node \(x\). Since initially every element in \(G\) has at most \(p\) pointers, it follows that the total amount of pointer addings in \(E\) is at least

\[
\frac{1}{2} \cdot (2\cdot 2^n)(q_1 - q_0) - n^p
\]

by using \(n \geq (1 - \alpha)2^n\), \(0 \leq \alpha \leq \frac{1}{2}\), and \(4p \leq q_1 - q_0\).

**Corollary 4.2.** Let \(G\) be a GU\((1, c, p)\) machine. Let \(TS\) be a complete 0-Turn sequence on \(n\) elements \((n\) is a power of two\). Suppose \(4p \leq a(1, n)\). Let \(E\) be an execution of TS on \(G\). Then at least \(\frac{1}{2} \cdot n \cdot a(1, n)\) pointer addings occur in \(E\).

We introduce some notions. An execution of an \(\alpha\)-Turn sequence TS on a GU\((i, c, p)\) machine is called conservative if the program for each Turn is minimal w.r.t. changes of contents of fields; i.e., the omission of one field change in the program for the Turn would yield that at the end of the Turn there would be no path from some element to its (new) set name. As a consequence, changes of the contents of fields from a pointer to \(n\)l do not occur in a conservative execution: all field changes are pointer addings.
Obviously, for each execution $E$ of an $x$-Turn sequence $TS$ on a GU$(i, c, p)$-machine $G$ there exists a conservative execution $E'$ of $TS$ on $G$ with cost not exceeding the cost of $E$ and that starts with the same initial contents of $G$. This is seen as follows:

- the initial contents for $E'$ equals that for $E$
- all creations of nodes performed by $E$ are also performed by $E'$
- the program for a Turn in $E'$ may (only) change the contents of a field into the contents that the field has at the end of the program for that Turn in $E$
- the program for a Turn in $E'$ is minimal w.r.t. changes of contents of fields: i.e., the omission of one field change in the program would yield that at the end of the Turn the same element would not have a path to its (new) set name in $G$.

Obviously, $E'$ is conservative and the cost of $E'$ does not exceed the cost of $E$. Therefore, it suffices to consider conservative executions only. We need the following claim.

**Claim 4.3.** Let $G$ be a GU$(i, c, p)$-machine. Let $TS$ be an $x$-Turn sequence. Suppose that the initial partition of $TS$ consists of sets of $x$-size $2^i$. Let $E$ be a conservative execution of $TS$ on $G$. Suppose that in the initial contents of $G$ for $E$ at most $b$ fields contain the same pointer to its (unique) set name in $G$. Consider execution $E'$ of $TS$ such that $G$ reflects a partition that consists of sets of $x$-size $2^i$, at most $F + 2 + e^{i-1}\cdot b$ fields contain the same pointer.

**Proof.** The bound trivially holds for $a = b = a$. Moreover, no fields contain pointers to nodes in layer $i$. Hence, we only need to consider pointers to nodes outside layer $i$. Suppose the bound holds for some $b$ with $b \geq a$. Then initially (if $b = a$) or after the execution of the Turn that yields sets of $x$-size $2^b$ (if $b > a$) at most $F + e^{i-1}\cdot 2^b + 1$ fields contain the same pointer in $G$. Consider $G$ at the end of the execution of the Turn yielding sets of $x$-size $2^b + 1$. Colour all fields with new pointers arisen from this Turn red. For any node $x$ outside layer $i$ there are at most $e^{i-1}$ set names that are reachable from node $x$, say that the collection of these set names is $S(x)$. Moreover, since the Turn sequence is executed conservatively, for every red field with a pointer to $x$ there exists some element $e$ for which all paths from $e$ to its (unique) set name in $S(x)$ use that red field. (Consequently, for distinct red fields with a pointer to $x$ such elements are distinct.) Since the sets arising from the Turn have size at most $2^b + 1$, there are at most $2^b + 1, e^{i-1}$ red fields with a pointer to $x$. Hence, at most $F + e^{i-1}(2^b + 1 + 2^b + 1) \leq F + e^{i-1}\cdot 2^{b+1}$ fields contain a pointer to $x$.

**Lemma 4.4.** Let $G$ be a GU$(i, c, p)$-machine for some $i > 1$. Let $TS$ be an $x$-Turn sequence $TS$ for some $x$, $0 \leq x \leq 2^{-(i-1)}$, and let $n$ be the number of elements. Suppose the initial partition of $TS$ consists of sets of $x$-size $A(i, q_0)$ and the resulting partition of $TS$ consists of sets of $x$-size $A(i, q_1)$. Let $E$ be an execution of $TS$ on $G$, where in the initial contents of $G$ at most $A(i, q_0 + 1)$ fields contain the same pointer. Suppose $e^{i-1} \cdot p \leq A(i, q_0), q_0 \geq 4$, and $q_1 - q_0 \geq 4$. Then at least $12^{i-1} n \cdot (q_1 - q_0)$ pointer addings occur in $E$.

**Proof.** We prove the lemma by induction. Let $i \geq 2$. Suppose that if $i - 1 \geq 2$ then the lemma holds for $i - 1$. We prove that the lemma holds for $i$. W.l.o.g. $E$ is conservative. Let $U$ be the universe of the elements in $TS$. There exists a universe $V \supseteq U$ with $|V| = 2^j$ and a 0-Turn sequence $TSO$ on $V$ such that $TS = TSO|_i$, and $n \geq (1-x)2^j$. We split $TS$ into consecutive subsequences $TS^{\text{pre}}, TS^{\text{cons}},$ and $TS_0, (0 \leq k \leq \lfloor (q_1 - q_0 - 1)/3 \rfloor - 1)$ such that $TS$ is a conservative, for every red field with a pointer to its (unique) set name in $G$. This is the restriction of $TS_0$ to $U_i$.

Consider execution $E$. Let $C_i$ be the contents of $G$ at the start of the execution of $TS_i$. Then $C_i$ represents the partition in sets of $x$-size $A(i, q_0 + 3k + 1)$. Since $E$ is conservative, it follows by Claim 4.3 that in $C_i$ the number of fields that contain the same pointer is at most

\[ A(i, q_0 + 1) + 2 \cdot e^{i-1} \cdot A(i, q_0 + 3k + 1) \]

\[ \leq A(i, q_0 + 3k + 1) + (1 + 2d(i, q_0)) \]

\[ \leq (A(i, q_0 + 3k + 1))^2 \]

\[ \leq A(i, q_0 + 3k + 2) \]

since initially in $G$ at most $A(i, q_0 + 1)$ fields contain the same pointer and since $e^{i-1} \cdot p \leq A(i, q_0), i \geq 2$, and $q_0 \geq 4$.

By Claim 4.5 (given below) it follows that at least $(1/2 \cdot 12^{i-1})n$ pointer addings occur in $E$ for the execution of $TS_i$. Hence, by $q_1 - q_0 \geq 4$ at least

\[ \frac{q_1 - q_0 - 1}{3} \left( \frac{1}{2 \cdot 12^{i-1}} n \right) \]

\[ \geq \frac{1}{2 \cdot 12^{i-1} \cdot 6} \cdot n \]

\[ \geq 12^{i-1} (q_1 - q_0) \cdot n \]

pointer addings occur during execution $E$ of $TS$.

We are left to prove Claim 4.5.

**Claim 4.5.** Let $0 \leq k \leq \lfloor (q_1 - q_0 - 1)/3 \rfloor - 1$. Let $A$ be an execution of $TS_0$ on $G$. Suppose that initially in $G$ (when the partition in sets of $x$-size $A(i, q_0 + 3k + 1)$ is reflected) at

File: 571J 138208. By:CV. Date:19/01/00. Time:16:01 LOPSM V8.0. Page 01/01
Codes: 6091 Signs: 4371. Length: 56 pic 0 pts, 236 mm
most \((A(i, q_0 + 3k + 1))^2 \leq A(i, q_0 + 3k + 2)\) fields contain
the same pointer (i.e., which designate the same node). Then
\(A\) contains at least \((1/2.12i - 1)n\) pointer addings.

Proof. W.l.o.g. \(A\) is conservative. (Note that every
change in a field of an element is a pointer adding now.)
Suppose \(A\) contains less than \((1/2.12i - 1)n\) pointer
addings. Let \(U\) be the collection of elements of which
the contents of the fields are not changed. Then \(U\) satisfies
\[
n' := |U| \geq \left(1 - \frac{1}{2.12i - 1}\right)n \geq \left(1 - \frac{1}{2.12^{i+1}}\right)
\]
for some \(\alpha'\) with \(0 \leq \alpha' \leq 2^{-i}\). Let \(TS'_k\) be given by
\(TS'_k = TS_{104 i}.\) Then \(TS'_k\) is an \(\alpha'\)-Turn sequence on
universe \(U'\), its initial partition consists of sets of \(\alpha'\)-size
\(A(i, q_0 + 3k + 1)\), and its result partition consists of sets of
\(\alpha'\)-size \(A(i, q_0 + 3k + 4)\).

We construct an execution \(A'\) of \(TS'_k\) on a
\(GU(i - 1, c, cp)\) machine \(G'\) by means of execution \(A\) of \(TS_k\)
as follows.

For each node \(y\) at layer \(i - 1\) or \(i\) of \(G\), we denote by
\(p_i(y)\) the contents of the \(h\)th field of \(y\). (Note that \(1 \leq h \leq c\)
for layer \(i - 1\) and \(1 \leq h \leq p\) for layer \(i\).) For each node \(y\) at
layer \(i - 1\) of \(G\), we denote by \(p_i(y)\) the contents of the \(h\)th
field of \(y\) (\(1 \leq h \leq cp\)). Then execution \(A'\) is obtained from \(A\)
by maintaining the following relations:

- The contents of \(G'\) are identical to the contents of \(G\)
with respect to layers \(0\) to \(i - 2\); i.e., the collections of nodes
in these layers are identical and the fields of these nodes
contain pointers to the same nodes (if any).
- Layer \(i - 1\) of \(G'\) consists of the elements of \(U'\) only;
these elements have \(cp\) pointer fields.
- For an element \(e \in U'\) in \(G'\), the contents of its fields
\(p_i(e)\) in \(G'\) (\(1 \leq h \leq cp\)) are given by
\[
p_i(e) = p_i(e)\) (\(1 \leq s \leq p, 1 \leq r \leq c\)
\]
which is \(nil\) if \(p_i(e) = nil\) (and which is the contents of the \(r\)th
field of the node pointed at by \(p_i(e)\) otherwise).

It is easily seen that initially and at each end of each Turn
there is a path from an element \(e \in U'\) to its set name \(s\) in \(G\)
iff there is a path from \(e\) to \(s\) in \(G'\). Therefore \(A'\) is an execution
on \(G'\) of the \(\alpha'\)-Turn sequence \(TS'_k\) on \(U'\).

By the condition given in the claim we have that initially
in \(G\) (when \(G\) reflects the initial partition in sets of \(\alpha'\)-size
\(A(i, q_0 + 3k + 1)\)) at most \((A(i, q_0 + 3k + 1))^2 \leq A(i, q_0 + 3k + 2)\)
fields contain the same pointer. Since the contents of
the fields of the elements in \(U'\) are not changed by \(A\) in \(G\),
this gives that execution \(A'\) on \(G'\) contains at most
\[
A(i, q_0 + 3k + 2).P
\]
pointer addings if \(P\) is the number of pointer addings
performed in \(A\). Moreover, it follows that initially at most
\[
(A(i, q_0 + 3k + 1))^4
\]
fields in \(G'\) contain the same pointer.

We show that the number of pointer addings in \(A'\) is at
least \(\lambda 2^{-i - 1} n. A(i, q_0 + 3k + 2)\). Let \(x\) and \(y\) be given by
\(x = A(i, q_0 + 3k)\) and \(y = A(i, q_0 + 3k + 3)\). Hence, by (1)
and \(i \geq 2\)
\[
A(i - 1, x) = A(i, q_0 + 3k + 1)
\]
\[
A(i - 1, y) = A(i, q_0 + 3k + 4).
\]
Note that by \(q_0 \geq 4\) and \(i \geq 2\) we have
\[
x = A(i, q_0 + 3k) \geq q_0 \geq 4,
\]
\[
y - x = A(i, q_0 + 3k + 3) - A(i, q_0 + 3k)
\]
\[
\geq A(i, q_0 + 3k + 2) \geq 4.
\]
Now we have that \(A'\) is an execution of the \(\alpha'\)-Turn
sequence \(TS'_k\) on the \(GU(i - 1, c, cp)\) machine \(G'\) with
\(0 \leq \alpha' \leq 2^{-i}\), that the initial partition of \(TS'_k\) consists of sets
of \(\alpha'\)-size \(A(i - 1, x)\), and that the result partition consists of
sets of \(\alpha'\)-size \(A(i - 1, y)\). We show that for \(i - 1 = 1\) and
\(i - 1 \geq 2\) the additional constraints for using Lemma 4.1 or
the induction hypothesis on \(A'\) are satisfied:

- \(i - 1 \geq 2\). Then by (12) we have that initially in \(G'\) at
most
\[
(A(i, q_0 + 3k + 1))^4 = (A(i - 1, x))^4 \leq A(i - 1, x + 1)
\]
fields contain the same pointer by using (13) and
\(A(i - 1, x + 1) = A(i - 2, A(i - 1, x)) \geq A(1, A(i - 1, x))\)
\[\leq 2^{4i - 1, x}\] which is the last inequality
follows with \(A(i - 1, x) \geq x \geq A(i, q_0) \geq A(3, 4) \geq 100\) (by
(15) and \(i - 1 \geq 2\)).

Note that since \(1 \leq c^{-i - 1}.p \leq A(i, q_0) \leq A(i - 1, x)\) holds
(viz., by the conditions of Lemma 4.4), we have
\[
1 \leq c^{-i - 2}.cp \leq A(i - 1, x)
\]
and that, by (16) and (15), we have
\[
y - x \geq 4 \wedge x \geq 4.
\]
By (17), (18), and (19), the induction hypothesis for \( i - 1 \) yields that there occur at least
\[
12^{-i+1} \cdot n . (y - x) \geq \frac{1}{2} 12^{-i} \cdot n . A(i, q_0 + 3k + 2)
\]
pointer additions in \( A' \) by using (10) and (16).

- If \( i = 1 \). Inequality (16) and the conditions in Lemma 4.4 give that there occur at least
\[
A(i, q_0 + 3k + 2) \geq 4 . A(i, q_0) \geq 4 . c . p .
\]
Hence,
\[
y - x \geq 4 . c . p .
\]
(20)

By (20) and Lemma 4.1 it follows that there occur at least
\[
\frac{1}{12^{i}} \cdot n . (y - x) \geq \frac{1}{12^{i}} \cdot n . A(i, q_0 + 3k + 2)
\]
pointer additions in \( A' \) by using (10) and (16).

By the above case analysis it follows that at least \( \frac{1}{12} \cdot 12^{-i+1} \cdot n . A(i, q_0 + 3k + 2) \) pointer additions occur in \( A' \).

By (11) it follows that at least \( \frac{1}{12} \cdot 12^{-i+1} \cdot n \) pointer additions in \( A \). This gives a contradiction with the assumption that there are less than \( \frac{1}{12} \cdot 12^{-i+1} \cdot n \) pointer additions. Thus, this proves Claim 4.5.

This concludes the proof of Lemma 4.4.

Lemma 4.4 yields the following result.

**Corollary 4.6.** Let \( G \) be a GU(i, c, c) machine for some \( i > 1 \). Let \( TS \) be a complete 0-Turn sequence and let \( n \) be the number of elements. Suppose \( c' \leq A(i, \lfloor \frac{1}{2} . a(i, n) \rfloor - 1) \) and \( a(i, n) \geq 10 \). Let \( E \) be an execution of \( TS \) on \( G \), where initially in \( G \) at most \( c' \) pointers contain the same pointer. Then at least \( \frac{1}{12} \cdot 12^{-i} \cdot n . a(i, n) \) pointer additions occur in \( E \).

**Proof.** W.l.o.g. \( E \) is conservative. Let \( q_0 = \lfloor \frac{1}{2} . a(i, n) \rfloor - 1 \) and \( q_1 = a(i, n) - 1 \). Then at the moment that \( G \) reflects the partition with sets of size \( A(i, q_0) = 2^k \) (for some \( b \)), it follows by Claim 4.3 that in \( G \) at most
\[
c'^{-1} + 2 . c'^{-1} . 2^k \leq A(i, q_0)(1 + 2 . A(i, q_0)) \leq A(i, q_0 + 1)
\]
fields contain the same pointer (by using \( i > 1 \) and \( q_0 \geq 4 \)). By Lemma 4.4 it follows that at least \( \frac{1}{12} \cdot 12^{-i} \cdot n . a(i, n) \) pointer additions occur in the part of execution \( E \) that corresponds to the subsequence of \( TS \) with the initial partition consisting of sets of size \( A(i, q_0) \) and resulting partition consisting of sets of size \( A(i, q_1) \). Thus the cost of \( E \) is at least
\[
\frac{1}{12^{i+1}} \cdot n . a(i, n).
\]

5. A GENERAL LOWER BOUND FOR THE UNION-FIND PROBLEM

**Lemma 5.1.** Let \( i \geq 1 \), \( c \geq 1 \). Let \( E \) be a UF(i, c)-execution of a complete 0-Turn sequence \( TS \) on \( n \) elements \( (n \text{ is a power of two}) \). Then \( E \) costs at least
\[
\frac{1}{4} \cdot 12^i \cdot i . (c + 1)^{-i} \cdot n . a(i, n) - (c + 1) \cdot n
\]
pointer additions if \( i \geq 2 \), \( a(i, n) \geq 10 \), and \( A(i, \lfloor \frac{1}{2} . a(i, n) \rfloor - 1) \geq (c + 1) \cdot n \), and it costs at least \( (1/12) \cdot n . a(i, n) \) pointer additions if \( i = 1 \) and \( a(i, n) \geq 4 . (c + 1) \).

**Proof.** Let \( C \) be the cost of \( E \). From Lemma 3.3 it follows that there exists an execution \( E' \) of \( TS \) on a GU(i, c + 1, c + 1) machine \( G \) with cost at most \( 2 \cdot i . (c + 1)^{-i} \cdot C \) if \( i > 1 \) and with cost at most \( C \) if \( i = 1 \), while initially at most \( 2 . (c + 1) \cdot n \) fields in \( G \) contain a pointer.

- For \( i = 1 \) Corollary 4.2 gives that the cost of execution \( E' \) is at least \( \frac{1}{12} \cdot n . a(i, n) \). Hence, \( C \geq \frac{1}{12} \cdot n . a(i, n) \).

- For \( i > 1 \) we change execution \( E' \) into execution \( E'' \) as follows. Consider the initial contents of \( G \) for \( E' \). Colour a minimal collection of fields red such that for each element in \( G \) there is a path to its set name using pointers in red fields only: i.e., if some red field would not be red, then there would be some element that would no longer have a path to its set name via only red fields. Colour all the other fields that contain a pointer blue. Now the (new) initial contents of \( G \) for \( E'' \) equals that for \( E' \), except that all the fields that are not red contain nil. Furthermore, execution \( E'' \) consists of first adding all the pointers in blue fields (at the beginning of the execution of the first operation) followed by \( E' \).

Hence, the cost of \( E'' \) is at most \( 2 \cdot i . (c + 1)^{-i} \cdot C + 2 . (c + 1) \cdot n \). Moreover, initially in \( G \) at most \( (c + 1) \cdot n \) fields contain the same pointer, since every set consists of one element and since the number of red fields is minimal (also cf. the proof of Claim 4.3). By Corollary 4.6 it follows that the cost of \( E'' \) is at least \( \frac{1}{12} \cdot 12^{-i} \cdot n . a(i, n) \), which establishes the result for \( C \).

This concludes the proof of Lemma 5.1.

**Lemma 5.2.** Let \( i \geq 1 \), \( c \geq 1 \), and \( n \geq 1 \). Let \( n \) and \( c \) satisfy \( n . a(n) \geq n . (c, c + 1) + 1 \). Let \( US \) be a sub-balanced Union sequence on a universe of \( n \) elements. Then every UF(i, c)-execution of \( US \) with \( 1 \leq i \leq n . a(n) - 3 \) costs at least \( n . a(i + 1, n) \) pointer additions.

**Proof.** Consider a UF(i, c)-execution \( E \) of \( US \). Let \( E \) have cost \( C \). Since \( US \) is sub-balanced, \( US \) consists of a balanced Union sequence \( US' \) on a sub-universe of \( n' = 2^k \geq 2n \) elements that is intermixed with additional Unions. We modify execution \( E \) into execution \( E' \) for \( US' \) as follows. For each Union \( Un \) in \( US' \) let Pre(\( Un \)) be the longest subsequence of \( US \) that ends with \( Un \) and that does not contain Unions of \( US' \) except for \( Un \). Then a program for a Union \( Un \) in \( US' \) consists of the sequence of instructions in \( E \) for the Unions in Pre(\( Un \)). In this way we obtain execution \( E' \) that obviously is a UF(i, c)-execution of \( US' \) with cost at most \( C \).
Let TS' be the 0-Turn sequence of which US' is an implementation. Then by Lemma 3.1 there exists a UF(i, c)-
equation E^* of TS' with cost at most C. We show that the cost of E^* is at least n.a(i + 1, n). First we need that by Lemma 2.5 we have
\[ a(i, n) \geq 8.12 \cdot i \cdot (c + 1)^{i-1} \cdot (2a(i + 1, n) + c + 1) \quad (21) \]

We show that E^* satisfies the conditions of Lemma 5.1.

- For i = 1 we have by (21) a(1, n) \geq 8.12 \cdot (2a(2, n) + c + 1) \geq 4(c + 1) + 1 and, hence, by n' > \frac{1}{2}n we have a(1, n') \geq a(1, n) - 1 \geq 4(c + 1).
- If i > 2 then we have (by n' > \frac{1}{2}n and by (21))
  \[ a(i, \lfloor \frac{1}{2}a(i, n') \rfloor - 1) \geq a(i, \lfloor \frac{1}{2}a(i, n) \rfloor - 2) \]
  \[ \geq a(i, 4.12 \cdot i \cdot (c + 1)^{i-1} \cdot (2a(i + 1, n) + c + 1) - 2) \]
  \[ \geq a(i, (c + 1)^i) \geq (c + 1)^i \]

and a(i, n') \geq \frac{1}{2}a(i, n) \geq 4.12 \cdot i \cdot (c + 1)^{i-1} \cdot (2a(i + 1, n) + c + 1) \geq 10.

Hence, by Lemma 5.1 the cost of E^* is at least
\[ \frac{1}{2}n', a(1, n'), \]
if i = 1, and at least
\[ \frac{1}{4.12} \cdot i \cdot (c + 1)^{i-1} \cdot a(i, n') - (c + 1).n', \]
if i > 1. By using a(i, n') \geq \frac{1}{2}a(i, n), n' > \frac{1}{2}n, and (21) this is at least n.a(i + 1, n). This concludes the proof of Lemma 5.2.

**Theorem 5.3.** There exists a constant d > 0 such that the following holds:

For any c-pointer machine, for any integer f and for any sub-balanced Union sequence on a universe of n elements there exists a Union–Find problem consisting of the Union sequence intermixed with f Find operations whose execution by the c-pointer machine has a cost that is at least d.f.a(x, f, n) if x(n, n) > x(c, c) + 1.

**Proof.** Let n and c satisfy the constraints given above. Consider some sub-balanced Union sequence US on n elements. Let
\[ i = \max \left\{ j \left| \left[ f \leq n.a(j, n) \cdot j \right. \right. \wedge \left. \left. 1 \leq j \leq x(n, n) - 2 \right] \lor j = 1 \right. \right. \}
\]
\[ (22) \]

We construct a Union–Find problem that contains US as the subsequence of Unions and that costs at least f.i and we show that i \geq \frac{1}{2}a(x, f, n). We distinguish two cases:

- i = 1. Then at any moment after the first Union, at least one element cannot equal its set name and, hence, any F finds performed on such elements cost at least f together.
- i > 1. We construct a Union–Find problem semi-online, starting from the (known) sequence US of Union operations and intermix it with Finds. If at some moment when some partition is reflected (i.e., initially or at the end of some operation) there is an element that has distance \geq i to its set name and if less than f Finds have been performed thus far, then perform a Find on that element. Otherwise perform the next Union or stop if a next Union does not exist. Let E be the execution of the Unions and Finds obtained in this way. We distinguish two cases:

  - At least f Finds have been performed. Then obviously these Finds have cost at least \geq f.i.
  - Less than f Finds have been performed. We change E into an execution E' of Union sequence US as follows. The initial contents of the pointer machine for execution E' is the contents for E at the beginning of the first Union. All Finds occurring before the first Union are ignored w.r.t. E'. (These Finds are condensed in the new initial contents of the pointer machine.) Furthermore, each execution of a Find (not occurring before the first Union) is appended to the execution of its previous Union. Then obviously the number of pointer addings in E' is at most that in E. Because less than f Finds have been performed, it follows that, initially and at the end of the (thus extended) execution of each Union, all elements have distance < i to their set names. Therefore, E' is a UF(i - 1, c)-execution of US with 1 \leq i - 1 \leq x(n, n) - 3. By Lemma 5.2 it follows that at least n.a(i, n) pointer addings occur in E'. Hence the cost of E is at least n.a(i, n).

Hence in both cases the cost is at least min\{f, i, n.a(i, n)\}. By i > 1 and (22) we have f, i \leq n.a(i, n). Hence the cost of E is at least f.i.

We now show that i \geq \frac{1}{2}a(x, f, n). We distinguish three cases:

- 1 \leq i < x(n, n) - 2. Then by (22) n.a(i + 1, n)/(i + 1) < f.
  - We certainly have by Lemma 2.8 n.a(i + 2, n) < f. By Lemma 2.4 it follows that i + 2 \geq a(f, n) and, hence, by i > 1 it follows that i \geq \frac{1}{2} \cdot (i + 2) > \frac{1}{2}a(x, f, n).
  - i = x(n, n) - 2 (and hence x(n, n) \geq 3). From a(f, n) \leq a(x, f, n) it follows that i = x(n, n) - 2 \geq \frac{1}{2} \cdot a(x, f, n).
  - i = 1 > x(n, n) - 2. Hence x(n, n) \leq 2. From a(f, n) \leq a(x, f, n) it follows that i = 1 \geq \frac{1}{2} \cdot a(x, f, n).

Combining the above cases gives that i \geq \frac{1}{2}a(x, f, n). By combining the above results it follows that the cost is at least \frac{1}{2}f.a(x, f, n).

**Theorem 5.3.** Theorem 5.3 implies that even if all Unions are known in advance, the worst-case time bound is still \Omega(f.a(x, f, n)) for all sub-balanced Union sequences on a pointer machine.
that are intermixed with $f$ appropriate Finds. Hence, the linear bound proved in [6] for Union–Find problems in which the structure of the (arbitrary) Union sequence is known in advance and that is implemented on a RAM does not extend to a pointer machine.

Finally, since each operation takes at least one step on a pointer machine, we obtain the following theorem.

**Theorem 5.4.** There exists a constant $d > 0$ such that the following holds:

For any $c$-pointer machine and for any $n$ and $f$ with $\alpha(n, n) > \alpha(c, c) + 1$ there is a Union–Find problem on $n$ elements with a sequence of $n - 1$ Union and $f$ Find operations whose execution by the $c$-pointer machine requires at least $d(n + f, \alpha, f, n)$ steps.

**Corollary 5.5.** For any pointer machine there exists a constant $d > 0$ such that for any $n > 1$ and $f > 0$ there is a Union–Find problem on $n$ elements with a sequence of $n - 1$ Union and $f$ Find operations whose execution by the pointer machine requires at least $d(n + f, \alpha, f, n)$ steps.

### 6. A GENERAL LOWER BOUND FOR THE SPLIT–FIND PROBLEM

We first describe the Split–Find problem on a pointer machine (see also [12]). Let $U$ be a linearly ordered collection of nodes, called elements. Suppose $U$ is partitioned into a collection of sets and suppose a (possibly new) unique node is related to each set, called set name. (For the regular Union and Find operations whose execution by the pointer machine requires at least $d(n + f, \alpha, f, n)$ steps.)

We want to be able to perform the following operations:

- **Split($A$, $B$).** Split the set $A \cup B$ with $A < B$ (i.e., $x < y$ for all $x \in A$, $y \in B$) and $A \not\subseteq B$ into the two new sets $A$ and $B$ (destroying the old set $A \cup B$) and relate set names to the resulting sets.

- **Find($x$).** Return the name of the current set in which element $x$ is contained.

As usual, the occurring set names must satisfy the condition that, at every moment, the names of the existing sets are distinct. Moreover, the operations are carried out semi-on-line: i.e., each operation must be completed before the next operation is known, while the subsequence of Splits may be known in advance. The definition and rules for pointer machine executions that solve the Split–Find problem are similar to that for the Union–Find problem as given in Subsection 2.1.

We use the results of Section 4 to obtain a lower bound for the Split–Find problem. Like in Section 3 we consider a Split sequence as a sequence of pairs $((A_x, B_x))_x$, where a pair $(A_x, B_x)$ represents the operation Split($A_x, B_x$). We define a sub-balanced Split sequence as a reversed sub-balanced Union sequence. Then obviously, there exists a sub-balanced Split sequence on every universe. A UF($i, c$)-execution of a Split sequence is defined similarly as for a Union sequence. We prove the equivalent of Lemma 5.2.

**Lemma 6.1.** Let $i \geq 1$, $c \geq 1$, and $n \geq 1$. Let $n$ and $c$ satisfy $\alpha(n, n) > \alpha(c, c) + 1$. Let $S$ be a sub-balanced Split sequence on a universe of $n$ elements. Then any UF($i, c$)-execution of $S$ with $1 \leq i \leq \alpha(n, n) - 3$ costs at least $n, a(i + 1, n)$ pointer addings.

**Proof.** Consider a UF($i, c$)-execution $E$ of Split sequence $S$ and let $C$ be the number of pointer addings in it. Let $G$ be the pointer machine on which $E$ is executed. Modify the execution such that no changes in fields from a pointer to $nil$ are performed and such that no creation of nodes occur in $E$ (which can easily be obtained by assuming that nodes that are created during $E$ exist in the initial contents of $G$ already, where they contain $nil$ in their fields at that moment). Obviously the thus modified execution $E$ is still an execution of $S$ and it contains exactly $C$ changes of field contents.

Let $S^{-1}$ be the reverse sequence of $S$. Then $S^{-1}$ is a sub-balanced Union sequence on universe $U$. We construct an execution $E'$ of $S^{-1}$ by means of execution $E$ of $S$ as follows. The initial contents of $G$ for $E'$ equals the final contents of $G$ after execution $E$ (i.e., the contents of $G$ at the moment that the program of the last operation in $S$ halts). Then $E'$ is obtained by maintaining the following relations with as few pointer addings as possible. At the end of an execution of a Union in $S^{-1}$, pointer machine $G$ has the same contents as at the beginning of the corresponding Split in $S$. Then apparently, a change of the contents of a field by $E'$ during the execution of a Union occurs only if there is a change of the contents of that field by $E$ during the corresponding Split in $S$. Hence, $E'$ contains at most $C$ changes of field contents.

Since at the beginning or at the end of every Union in $S^{-1}$ the contents of the pointer machine is identical to that at the end or at the beginning of the corresponding Split in $S$, it follows that $E'$ is a UF($i, c$)-execution of Union sequence $S^{-1}$. Lemma 5.2 yields that $C \geq n, a(i + 1, n)$.

Completely similar to the proof of Theorem 5.3 we can prove the following theorem.

**Theorem 6.2.** There exists a constant $d > 0$ such that the following holds:

For any $c$-pointer machine, for any integer $f$ and for any sub-balanced Split sequence on a universe of $n$ elements there exists a Split–Find problem consisting of the Split sequence intermixed with $f$ Find operations whose execution by the $c$-pointer machine has a cost that is at least $d, f, \alpha(f, n)$ if $\alpha(n, n) > \alpha(c, c) + 1$.

Theorem 6.2 implies that even if all Splits are known in advance, the worst-case time bound on a pointer machine is still $\Omega(f, \alpha(f, n))$ for all sub-balanced Split sequences that
are intermixed with appropriate Finds. Hence, the linear bound proved in [6] for Split–Find problems on a RAM does not extend to a pointer machine, even if the sequence of Splits is known in advance.

Like for the Union Find Problem we obtain the following theorems.

**THEOREM 6.3.** There exists a constant $d > 0$ such that the following holds:

For any $c$-pointer machine and for any $n$ and $f$ with $\alpha(n, n) > \alpha(c, c) + 1$ there is a Split–Find problem on $n$ elements with a sequence of $n-1$ Split and $f$ Find operations whose execution by the $c$-pointer machine requires at least $d \cdot (n + f \cdot \alpha(f, n))$ steps.

**COROLLARY 6.4.** For any pointer machine there exists a constant $d > 0$ such that for any $n > 1$ and $f \geq 0$ there is a Split–Find problem on $n$ elements with a sequence of $n-1$ Split and $f$ Find operations whose execution by the pointer machine requires at least $d \cdot (n + f \cdot \alpha(f, n))$ steps.

Finally, we make some remarks about the separation condition for the Split–Find problem. In case the separation condition holds, the lower bound of Theorem 6.3 becomes valid for a uniform $d$ for all $n$ independent of $c$. (However, in this case we need to include all changes of contents of fields in our ultimate complexity measure, i.e., including changes to nil.) This matches the result in [15] for the Union–Find problem with the separation condition. We will not present the proof here.

7. FINAL REMARKS

We want to remark that even if during a Union or a Split the new set name is not assigned to the resulting set immediately, but it is assigned to it at some later time before or during the first Find that is performed on an element of that set, we still can prove Theorems 5.4 and 6.3, respectively. We will not present the proof here.

ACKNOWLEDGMENTS

The author thanks the anonymous referees for the useful comments.

REFERENCES