Lower Bounds for the Union-Find and the Split-Find Problem on Pointer Machines

J.A. La Poutré

RUU-CS-89-21
October 1989

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Lower Bounds for the Union-Find and the Split-Find Problem on Pointer Machines

J.A. La Poutré

Technical Report RUU-CS-89-21
October 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

Lower Bounds for the Union-Find and the Split-Find Problem on Pointer Machines*

J.A. La Poutré

Department of Computer Science, University of Utrecht,

P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

October 1989

Abstract

A well-known result of Tarjan (cf. [15]) states that for all n and $m \geq n$ there exists a sequence of $n-1$ Union and m Find operations that needs at least $\Omega(m.\alpha(m,n))$ execution steps on a pointer machine that satisfies the separation condition. In [1, 16] the bound was extended to $\Omega(n+n.\alpha(m,n))$ for all m and n. In this paper we prove that this bound holds on a general pointer machine without the separation condition and we prove that the same bound holds for the Split-Find problem as well.

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 "Union-Find problem".

*This research was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

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 $O(m.\alpha(m,n))$ for $n-1$ unions and $m \geq n$ finds, where α is the inverse Ackermann function. The algorithm can be 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 known Union-Find algorithms that run on pointer machines in the above time and that use a form of path compaction [16]. In [10] a new algorithm without path compaction is presented that runs on a pointer machine and that has a worst-case time bound of $O(\alpha(f,n))$ for the f^{th} Find, within the bound of $O(n+m.\alpha(m,n))$ for m Finds on n elements as a whole.

In 1979 (cf. [15]) Tarjan proved a lower bound on the time complexity of executions of the Union-Find problem 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 $\Omega(m.\alpha(m,n))$ time, if $m \geq n$. In [1, 16] the bound was extended to $\Omega(n+m.\alpha(m,n))$ 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 can be 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 $\Omega(n+m.\alpha(m,n))$ 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) singleton sets and suppose we want to be able to perform the following operations:

- Split(x): split the set in which x is contained into two sets, one set consisting of all elements in the set $\geq x$ 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.

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 $O(n + m.\log^* n)$ 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 $O(n + m.\alpha(m,n))$ time on a pointer machine. Until now no lower bound was found for the Split-Find problem on a pointer machine.

We prove a $\Omega(n + m.\alpha(m,n))$ 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 that 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 although the Split-Find problem can be solved in linear time on a RAM (cf. [6]), this is not possible on a pointer machine.

Table 1: Complexity on Pointer Machines

Problem[1]	General model		Separation condition	
UNION-FIND				
worst case/instruction	$O(\log\log n)$	[3][2]	$\Theta(\frac{\log n}{\log\log n})$	[2]
amortized	$\Theta(n + m.\alpha(m,n))$	**new**	$\Theta(n + m.\alpha(m,n))$	[15]
SPLIT-FIND				
worst case/instruction	$\Theta(\log\log n)$	[12]	$\Theta(\log n)$	[12]
amortized	$\Theta(n + m.\alpha(m,n))$	**new**	$\Theta(n + m.\alpha(m,n))$	**new**

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, but 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

[1] n is the number of elements and m is the number of Finds

[2] for special cases of the Union-Find problem

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 1 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 solves the problem in linear time. (Viz., take a pointer machine with at most 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 α-bound should hold. In this paper we prove that this bounds holds indeed, and moreover we show that there is a uniform constant d that holds for *all* pointer machines, such that a lower bound of $d.(n + \alpha(m, n))$ steps holds for all m and *asymptotically* for 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 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). Each node contains c fields that each may contain 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 restrict the way of addressing yet. The special way of addressing will be condensed 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 minimal length of any path from x to y.

The *Union-Find problem* on a pointer machine can be formulated as follows (also cf. [15] or [12, 14, 16]). Let U be a collection of nodes, called elements. Suppose U is partitioned into a collection of sets and suppose to each set a (possibly new) unique node is 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:

- 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
- Find(x): return the name of the current 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. 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 Unions may be known in advance.
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, 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 *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 that field into some pointer (hence, not *nil*).
- the cost of 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) 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.) 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 *nil*.

2.2 The Ackermann function

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

$$\begin{array}{rcll} A(0,x) & = & 2x & \text{for } x \geq 0 \\ A(i,0) & = & 1 & \text{for } i \geq 1 \\ A(i,x) & = & A(i-1, A(i,x-1)) & \text{for } i \geq 1,\ x \geq 1. \end{array} \tag{1}$$

The row inverse a of A and the functional inverse α of A are defined by

$$a(i,n) = min\{x \geq 0 | A(i,x) \geq n\} \qquad (i \geq 0,\ n \geq 0) \tag{2}$$

$$\alpha(m,n) = min\{i \geq 1 | A(i, 4\lceil m/n \rceil) \geq n\} \qquad (m \geq 0,\ n \geq 1) \tag{3}$$

Here we take $\lceil 0 \rceil = 1$. (Note that $\alpha(0,n) = \alpha(n,n)$.) 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, 15, 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 $A(i,1) = 2$, $A(i,2) = 4$, $A(i+1,3) = A(i,4)$ and $A(i+1,4) = A(i, A(i,4))$ for $i \geq 0$.

Lemma 2.1

$$\begin{array}{rcll} A(i',x') & \geq & A(i,x) & (i' \geq i,\ x' \geq x) \\ a(i,n) & \leq & a(i',n') & (i \geq i',\ n' \geq n) \\ \alpha(m',n') & \leq & \alpha(m,n) & (m' \geq m,\ n' \leq n) \end{array}$$

Lemma 2.2

$$\begin{array}{lcll} a(i,A(i,x)) & = & x & (i \geq 0,\ x \geq 0) \\ a(i,A(i+1,x+1)) & = & A(i+1,x) & (i \geq 0,\ x \geq 0) \\ a(i,n) & = & a(i,a(i-1,n))+1 & (i \geq 1,\ n \geq 2) \end{array}$$

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 $a(i,n) \geq 1$ and by (2), (1) and $i \geq 1$ we find

$$\begin{array}{l} a(i,n) = \\ \quad = min\{j \geq 1 | A(i,j) \geq n\} \\ \quad = min\{j \geq 1 | A(i-1,A(i,j-1)) \geq n\} \\ \quad = min\{j \geq 1 | A(i,j-1) \geq a(i-1,n)\} \\ \quad = min\{j' \geq 0 | A(i,j')) \geq a(i-1,n)\} + 1 \\ \quad = a(i,a(i-1,n)) + 1. \end{array}$$

□

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

Let $a^{(0)}(i,n) := n$ *and* $a^{(j+1)}(i,n) := a(i,a^{(j)}(i,n))$ *for* $i,j \geq 0,\ n \geq 1$. *Then* $a(i,n) = min\{j | a^{(j)}(i-1,n) = 1\}$ *for* $i,n \geq 1$.

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

$$\begin{array}{rcl} A(0,x) & = & 2x \\ A(1,x) & = & 2^x \\ A(2,x) & = & \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{x \text{ two's}} \\ A(3,x) & = & \underbrace{2^{2^{2^{\cdot^{\cdot^{2}}}}} \Big\} 2^{2^{\cdot^{\cdot^{2}}}} \Big\} \cdots {}^{2^{2}}2\}2\}1 \text{ two}}_{x \text{ braces}} \text{ two's} \text{ two's} \text{ two's}. \end{array}$$

On the other hand we have for $n \geq 1$:

$$\begin{array}{lcll} a(0,n) & = & \lceil \frac{n}{2} \rceil & \\ a(1,n) & = & \lceil \log n \rceil & = min\{j | \lceil \frac{n}{2^j} \rceil = 1\} \\ a(2,n) & = & \log^* n & = min\{j | \lceil \log^{(j)} n \rceil = 1\} \\ a(3,n) & = & & min\{j | \log^{*(j)} n = 1\} \end{array}$$

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 α as follows.

Lemma 2.4 $\alpha(m,n) = min\{i \geq 1 | a(i,n) \leq 4.\lceil m/n \rceil\}$.

We state some lemma's that we will need in the sequel. The proofs can be skipped at first reading. The lemma's use the following:

$$n \geq 3 \wedge i \geq 1 \Rightarrow a(i,n) > A(i+1, a(i+1,n) - 2). \tag{4}$$

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 $\alpha(n,n) > \alpha(c,c) + 1$ the following holds for i with $1 \leq i \leq \alpha(n,n) - 3$:*

$$a(i,n) \geq 8.12^i.i.(c+1)^{i-1}.(\,2a(i+1,n) + c + 1).$$

Proof. Let n and c satisfy $\alpha(n,n) \geq \alpha(c,c) + 2$. (Hence $\alpha(n,n) \geq 3$.)

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

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

$$n > A(\alpha(n,n) - 1, 4) = A(\alpha(n,n) - 2, A(\alpha(n,n) - 2, 4))) \geq A(\alpha(n,n) - 2, c).$$

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

Claim 2.7 $A(i+1, x-2) \geq 2.12^{i+1}.i.x^i$ *for $x \geq 6$ and $i \geq 1$.*

Proof. We prove the claim by induction to i and x. Firstly, For $i = 1$ and $x = 6$ we have $A(2,4) = A(1, A(1,4)) = 2^{16} \geq 2.12^2.1.6$. For $i > 1$ and $x = 6$ we have by induction

$$A(i+1,4) = A(i, A(i,4)) \geq 2^{A(i,4)} \geq 2^{2.12^i.(i-1).6^{i-1}} \geq 2.12^{i+1}.i.6^i.$$

Finally, for $i > 1$ and $x > 6$ we have by induction

$$A(i+1, x-2) = A(i, A(i+1, x-3)) \geq 2^{A(i+1,x-3)} \geq 2^{2.12^{i+1}.i.(x-1)^i} \geq 2.12^{i+1}.i.x^i.$$

This proves the claim. □

Let i be such that $1 \leq i \leq \alpha(n,n) - 3$. Note that $i+2 \leq \alpha(n,n) - 1$ 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, Equation (4) and Claim 2.7 give that

$$a(i,n) > 2.12^{i+1}.i.(a(i+1,n))^i = 8.12^i.i.(a(i+1,n))^{i-1}(2a(i+1,n) + a(i+1,n)).$$

By Claim 2.6 the inequality of Lemma 2.5 follows. □

Lemma 2.8 *Let $n \geq 0$, $1 \leq i \leq \alpha(n,n) - 2$. Then $a(i+1,n) < \frac{a(i,n)}{i}$.*

Proof. Since $i+2 \leq \alpha(n,n)$ we have $a(i+1,n) \geq 5$ and $n \geq 3$. Claim 2.7 gives that $A(i+1, x-2) \geq i.x$ for $x \geq 6$ and $i \geq 1$. Moreover, $A(i+1, 5-2) = A(i,4) \geq i.5$ by Claim 2.7 or 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 operation. 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 $((A_k, B_k))_k$ to denote the Union sequence US. US is called 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^a (for some integer a). A *Union Turn* or *0-Turn* T with initial partition P is a 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^{a+1}). A *0-Turn sequence* $TS = (T_i)_i$ 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 α, $0 \leq \alpha < \frac{1}{2}$, with $|U| \geq (1-\alpha).|V|$. 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 α-Turn or 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 $\{A \cup B | (A, B) \in T|_U \wedge A \cup B \neq \emptyset\}$. We call the sets in such a partition of U to have *α-size* 2^a if the sets in the corresponding partition of V have size 2^a. (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 $0 \leq \alpha < \frac{1}{2}$ the partition consists of sets of size $\leq 2^a$ of which at least one must have size $> 2^{a-1}$.) The sequence $(T_i|_U)_i$ is called an *α-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 partition and the final partition are completely determined by the α-Turn sequence. A 0-Turn sequence is *complete* if the initial partition consists of singleton sets and the result partition consists of one set.

The *operation* α-Turn T is given by: for each pair $(A, B) \in T$ $(A \neq \emptyset \vee B \neq \emptyset)$, join the sets A and B (while destroying these sets if both A and B are not 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.) The names of the resulting sets need to be distinct.

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 may be known in advance (and hence because of the semi on-line condition, the entire Union sequence may be known in advance). An *execution* of an α-Turn Sequence on a pointer machine consists of a (so-called initial) contents of the pointer machine together with a sequence of executions of α-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 α-Turn is carried out by executing a program, which halts having modified the contents of the pointer machine to reflect the α-Turn and hence to reflect the new partition of the universe

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 subsequence 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 $|V| > \frac{1}{2}n$ 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 α-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 i 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 $j \geq 1$).*

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 $2.(c+1)^i.n$ fields that contain a pointer, and such that EE has cost that is at most $2.i.(c+1)^{i-1} \cdot C$ if $i \geq 2$ and at most C if $i = 1$.*

Proof. Let G be a c-pointer machine G on which execution E is performed. Let the 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 j $(0 \leq j \leq i)$ 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_j in GG' with $1 \leq j \leq i$ contains a pointer to the representative x_{j-1} in its 0^{th} field;
- if in G node x contains a pointer to node y $(1 \leq a \leq c)$ in its a^{th} field, 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 and 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 GG'.

We describe how to obtain an execution EE on GG. Each node x' in GG' has at most one representative node x in GG and conversely, each node in GG is the representative of precisely 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 consists of those nodes x for which node x' in the initial contents of GG' is reachable from some element in GG'.
- 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 node x (if present) contains a pointer to node y if y exists and its 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. Moreover, a node in layer j: $0 < j < i$ of GG' has at most $\sum_{k=j}^{1}(c+1)^{j-k} \leq 2.(c+1)^{j-1} - 1 \leq 2(c+1)^{i-2} - 1$ nodes outside layer 0 of GG' that are reachable from it. Finally, each field in G corresponds to at most i fields in GG. Therefore it follows that any pointer adding in G can certainly be performed within a factor $i.((c+1).(2.(c+1)^{i-2} - 1)) + 1 \leq 2.i.(c+1)^{i-1}$ of cost if $i \geq 2$. (For, any node that becomes reachable has $c+1$ fields that may contain a pointer (at a cost of 1 per pointer) except for the nodes in layer 0 that contain *nil* in their fields only (having cost 0).) For $i = 1$ we obtain factor 1, since no node outside layer 0 can become reachable from an element during the execution of TS.

Finally it is easily seen that initially in GG there are at most $2.(c+1)^i.n$ fields that do not contain *nil* (and even $(c+1).n$ for $i = 1$). □

4 Lower bounds on GU(i, c, p) machines

In this section we will prove lower bounds for GU(i, c, p) machines.

Lemma 4.1 *Let G be a GU$(1, c, p)$ machine. Let TS be an α-Turn sequence for some α, $0 \leq \alpha \leq \frac{1}{4}$, and let n be the number of elements. Suppose the initial partition consists of sets of α-size 2^{q_0} and the result partition consists of sets of α-size 2^{q_1}. Let $q_1 - q_0 \geq 4p$. Let E be an execution of TS on G. Then at least $\frac{1}{12}.n.(q_1 - q_0)$ pointer addings occur in E.*

Proof. Let U be the universe of elements of TS. By the definition of α-Turn, there exists a universe $V \supseteq U$ and a (original) 0-Turn sequence TSO on V such that $TS = TSO|_U$. Let integer v be given by $|V| = 2^v$.

Consider an execution EE of TSO on G. For each 0-Turn T in TSO we define a so-called matching sequence in the following way: the matching sequence for T contains all the pairs (e, s) of elements e and set names s such that s is the set name for e at the end of the Turn. Now consider the sequence obtained by concatenating the matching sequences of the 0-Turns in the right order. Then it obviously consists of $(q_1 - q_0).2^v$ pairs. For some node s that occurs as a set name in the sequence, consider the last time that s is the name of some set in the sequence. Let this set be set A. Suppose A has 2^a elements. Since at the end of a 0-Turn the set names of distinct sets must be different, it follows that for all 0-Turns preceding to the 0-Turn yielding sets of size 2^a at most one set per 0-Turn has s as its set name. Therefore, at most $1 + 2 + 2^2 + \ldots + 2^{a-1} = 2^a - 1$ different elements have had s as their set name before set A occurred. These elements may be elements of A. Therefore at least half the number of pairs of elements and set names occurring in the matching sequence are distinct. Hence the number of distinct pairs is at least $\frac{1}{2}.(q_1 - q_0).2^v$.

Hence, any execution of TSO contains at least

$$\frac{1}{2}.(q_1 - q_0).2^v \tag{5}$$

different pairs in its matching sequence.

Now consider execution E of TS. Note that E can be augmented to be an execution of TSO by performing at most $(q_1 - q_0).(2^v - n)$ pointer addings in elements of $V \backslash U$ during the Turns, yielding at most that many pairs in the matching sequence. By (5) this gives that there must be at least $\frac{1}{2}.(q_1 - q_0).(2n - 2^v)$ different pairs in the matching sequence of the E.

Note that each pair (x, s) in the matching sequence 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 is at least

$$\begin{aligned}
&\frac{1}{2}(2n - 2^v)(q_1 - q_0) - n.p \\
&\geq \frac{1}{2}(2n - \frac{1}{1-\alpha}.n)(q_1 - q_0) - n.p \\
&\geq \frac{1}{2}(2n - (\frac{4}{3}\alpha + 1).n)(q_1 - q_0) - n.p \\
&= (\frac{1}{2} - \frac{2}{3}\alpha).n.(q_1 - q_0) - n.p \\
&\geq (\frac{1}{4} - \frac{2}{3}\alpha).n.(q_1 - q_0) \\
&\geq \frac{1}{12}.n.(q_1 - q_0).
\end{aligned}$$

by using $0 \leq \alpha \leq \frac{1}{4}$ 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 $\frac{1}{12}.n.a(1, n)$ pointer addings occur in E.*

We introduce some notions. An execution of an α-Turn sequence TS on a GU(i, c, p) machine is called *conservative* if the execution of each Turn contains a minimal number of changes of contents of fields: the omission of one field change in 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 *nil* do not occur in a conservative execution: all field changes are pointer addings.

Obviously, for each execution E of an α-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 change the contents of a field to the contents of that field at the end of the program for that Turn in E.
- during each Turn only a minimal number of changes of the contents of fields is performed by E': i.e., the omission of one change would yield that some element would not have a path to its (new) set name after the Turn 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 α-Turn sequence. Suppose the initial partition of TS consists of sets of α-size 2^a. Let E be a conservative execution of TS on G. Suppose that in the initial contents of G for E at most F fields contain the same pointer. Then at the moment in E that G reflects a partition that consists of sets of α-size 2^b, at most $F+2.c^{i-1}2^b$ fields contain the same pointer.*

Proof. The bound trivially holds and for $a = b$. Moreover, no fields contain pointers to elements since elements are in layer i. Now consider nodes that are not elements. 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 α-size 2^b (if $b > a$) at most $F + c^{i-1}2^{b+1}$ fields contain the same pointer in G. Consider G at the end of the execution of the (next) Turn yielding sets of α-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 c^{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 $2^{b+1} \cdot c^{i-1}$ red fields with a pointer to x. Hence, at most $F + c^{i-1}(2^{b+1} + 2^{b+1}) \leq F + c^{i-1}2^{b+2}$ 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 α-Turn sequence TS for some α, $0 \leq \alpha \leq 2^{-(i+1)}$, and let n be the number of elements. Suppose the initial partition of TS consists of sets of α-size $A(i,q_0)$ and the resulting partition of TS consists of sets of α-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. Finally, let $c^{i-1} \cdot p \leq A(i,q_0)$, $q_0 \geq 4$ and $q_1 - q_0 \geq 4$. Then at least $12^{-i}.n.(q_1 - q_0)$ pointer addings occur in E.*

Proof. We prove the lemma for by induction. Let $i \geq 2$. Suppose that the lemma holds for $i-1$ if $i - 1 \geq 2$. 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 with $|V| = 2^v$ and a 0-Turn sequence TSO such that $TSO|_U = TS$ and $n \geq (1-\alpha)2^v$. We split TS into consecutive subsequences TS^{pre}, TS^{post} and TS_k $(0 \leq k \leq \lfloor \frac{q_1 - q_0 - 1}{3} \rfloor - 1)$ such that $TS = TS^{pre}, (TS_k)_k, TS^{post}$ and for each k, the initial partition of TS_k consists of sets of α-size $A(i, q_0 + 3k + 1)$ and the result partition consists of sets of α-size $A(i, q_0 + 3k + 4)$. (The subsequence TS^{post} may be empty.) Let TSO be split into subsequences TSO^{pre}, TSO^{post} and TSO_k such that $TSO_k|_U = TS_k$. (Obviously TS_k is an α-Turn sequence that is the restriction of TSO_k to U.)

Consider execution E. Let C_k be the contents of G at the start of the execution of TS_k. Then C_k represents the partition in sets of α-size $A(i, q_0 + 3k + 1)$. Since E

is conservative, it follows by Claim 4.3 that in C_k the number of fields that contain the same pointer is at most

$$\begin{aligned} & A(i, q_0+1) + 2 \cdot c^{i-1} \cdot A(i, q_0+3k+1) && (6)\\ \leq\ & A(i, q_0+3k+1) \cdot (1 + 2A(i, q_0)) && (7)\\ \leq\ & (A(i, q_0+3k+1))^2 && (8)\\ \leq\ & A(i, q_0+3k+2) && (9) \end{aligned}$$

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

By Claim 4.5 (given below) it follows that at least $\frac{1}{2.12^{i-1}}n$ pointer addings occur in E for the execution of TS_k. Hence, by $q_1 - q_0 \geq 4$ at least

$$\lfloor \frac{q_1 - q_0 - 1}{3} \rfloor . (\frac{1}{2.12^{i-1}}.n) \geq \frac{1}{2.12^{i-1}} . \frac{q_1 - q_0}{6} . n \geq \frac{1}{12^i} . (q_1 - q_0).n$$

pointer addings occur during execution E of TS.

We are left to prove Claim 4.5.

Claim 4.5 *Let $0 \leq k \leq \lfloor \frac{q_1 - q_0 - 1}{3} \rfloor - 1$. Let A be execution of TS_k on G. Suppose that initially (when the partition in sets of α-size $A(i, q_0 + 3k + 1)$ is reflected) at most $A(i, q_0 + 3k + 1))^2 \leq A(i, q_0 + 3k + 2)$ fields contain the same pointer. Then A contains at least $\frac{1}{2.12^{i-1}}n$ 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 $\frac{1}{2.12^{i-1}}n$ pointer addings. Let U' be the collection of elements of which the contents of the fields are not changed. Then U' satisfies

$$n' := |U'| \geq (1 - \frac{1}{2.12^{i-1}})n \geq (1 - \frac{1}{2.12^{i-1}}).(1 - \frac{1}{2^{i+1}}).2^v = (1 - \alpha').2^v \qquad (10)$$

for some α' with $0 \leq \alpha' \leq 2^{-i}$. Let TS'_k be given by $TS'_k = TSO_k|_{U'}$. Then TS'_k is an α'-Turn sequence on universe U'.

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

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

- the contents of G' is identical to the contents of G with respect to layers 0 to $i-2$: i.e., the collection 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 $p'_h(e)$ in G' $(1 \leq h \leq cp)$ are given by $p'_{(l-1)c+k}(e) = p_k(p_l(e))$ $(1 \leq l \leq p,\ 1 \leq k \leq c)$, which is nil if $p_l(e) = nil$ (and which is the contents of the k^{th} field of the node pointed at by $p_l(e)$ otherwise).

Since in G the fields of elements $e \in U'$ are not changed during execution A, it is easily seen that at any time 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 α'-Turn sequence TS'_k on U'.

By the condition given in the claim we have that initially in G (when G reflect the initial partition in sets of α-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

$$A(i, q_0 + 3k + 2) \cdot P \tag{11}$$

pointer addings if P is the number of pointer addings performed in A. Moreover, it follows that initially at most

$$(A(i, q_0 + 3k + 1))^4 \tag{12}$$

fields in G' contain the same pointer.

We show that the number of pointer addings in A' is at least $\frac{1}{2}.12^{-(i-1)}.n.A(i, q_0 + 3k + 2)$.

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 $i \geq 2$

$$A(i-1, x) = A(i, q_0 + 3k + 1) \tag{13}$$
$$A(i-1, y) = A(i, q_0 + 3k + 4). \tag{14}$$

Note that by $q_0 \geq 4$ and $i \geq 2$ we have

$$x = A(i, q_0 + 3k) \geq q_0 \geq 4 \tag{15}$$
$$y - x = A(i, q_0 + 3k + 3) - A(i, q_0 + 3k) \geq A(i, q_0 + 3k + 2) \geq 4. \tag{16}$$

Now we have that A' is an execution of the α'-Turn sequence TS'_k on the GU$(i-1, c, cp)$ machine G' with $0 \leq \alpha' \leq 2^{-i}$, and that the initial partition of TS'_k consists of sets of α'-size $A(i-1, x)$ and the result partition consists of sets of α'-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

$$(A(i, q_0 + 3k + 1))^4 = (A(i-1, x))^4 \leq A(i-1, x+1) \tag{17}$$

fields contain the same pointer by using (13) and $A(i-1, x+1) = A(i-2, A(i-1, x)) \geq A(1, A(i-1, x)) = 2^{A(i-1,x)} \geq (A(i-1, x))^4$ where 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 $1 \leq c^{i-1} \cdot p \leq A(i, q_0) \leq A(i-1, x)$ holds (viz. the conditions of Lemma 4.4) we have

$$1 \leq c^{i-2} \cdot (cp) \leq A(i-1, x) \tag{18}$$

and that by (16) and (15) we have

$$y - x \geq 4 \wedge x \geq 4. \tag{19}$$

By (17), (18) and (19), the induction hypothesis for $i-1$ yields that there occur at least

$$12^{-(i-1)}.n'.(y-x) \geq \frac{1}{2.12^{i-1}}.n.A(i, q_0 + 3k + 2)$$

pointer addings in A' by using (10) and (16).

- $i - 1 = 1$. Unequality (16) and the conditions in Lemma 4.4 give $y - x \geq A(i, q_0 + 3k + 2) \geq 4.A(i, q_0) \geq 4 \cdot cp$. Hence

$$y - x \geq 4cp \tag{20}$$

By (20) and Lemma 4.1 it follows that there occur at least

$$\frac{1}{12}.n'.(y-x) \geq \frac{1}{12} \cdot \frac{1}{2}.n.A(i, q_0 + 3k + 2)$$

pointer addings in A' by using (16).

By the above case analysis it follows that at least $\frac{1}{2}.12^{-(i-1)}.n.A(i, q_0 + 3k + 2)$ pointer addings occur in A'. By (11) it follows that there are at least $\frac{1}{2}.12^{-(i-1)}.n$ pointer addings in A. Contradiction with the assumption that there are less than $\frac{1}{2}.12^{-(i-1)}.n$ pointer addings. 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 $c^i \leq A(i, \lfloor \frac{1}{2}.a(i,n) \rfloor - 1)$ and $a(i,n) \geq 10$. Let E be an execution of TS on G, where initially in G at most c^{i-1} fields contain the same pointer. Then at least $\frac{1}{2}.12^{-i}.n.a(i,n)$ pointer addings 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 that G reflects the partition with sets of α-size $A(i,q_0) =: 2^b$, it follows by Claim 4.3 that in G at most

$$c^{i-1} + c^{i-1}2^{b+1} \leq A(i,q_0).(1 + 2.A(i,q_0)) \leq A(i,q_0+1)$$

fields contain the same pointer (by using $i > 1$ and $q_0 \geq 4$). By Lemma 4.4 it follows that at least $\frac{1}{2}.12^{-i}.n.a(i,n)$ pointer addings 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 $\frac{1}{2}.12^{-i}.n.a(i,n)$. □

5 The general lower bound for the Union-Find problem

Claim 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 is a power of two).*
Then E costs at least $\frac{1}{4.12^i.i.(c+1)^{i-1}}.n.a(i,n) - (c+1).n$ pointer addings if $i \geq 2$, $a(i,n) \geq 10$ and $A(i, \lfloor \frac{1}{2}a(i,n) \rfloor - 1) \geq (c+1)^i$, and it costs at least $\frac{1}{12}.n.a(1,n)$ pointer addings 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 $2.i.(c+1)^{i-1}.C$ if $i > 1$ and with cost at most C if $i = 1$, while initially at most $2.(c+1)^i.n$ fields in G contain a pointer.

- For $i = 1$ Corollary 4.2 gives that the cost of execution E' is at least $\frac{1}{12}.n.a(1,n)$. Hence, $C \geq \frac{1}{12}.n.a(1,n)$.
- 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 not have a path to its set name via red fields only any more. Colour all other fields that contain a pointer blue. Now the (new) initial contents of G for E'' equals that for E' except that all fields that are not red contain *nil*. Furthermore, execution E'' consists of first adding all 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 $2.i.(c+1)^{i-1}.C + 2.(c+1)^i.n$. Moreover, initially in G at most $(c+1)^{i-1}$ 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 $\frac{1}{2}.12^{-i}.n.a(i,n)$, what establishes the result for C.

This concludes the proof of Claim 5.1. □

Lemma 5.2 *Let $i \geq 1$, $c \geq 1$ and $n \geq 1$. Let n and c satisfy $\alpha(n,n) > \alpha(c,c)+1$. Let US be a sub-balanced Union sequence on a universe of n elements. Then any UF(i,c)-execution of US 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 US. Let E have cost C. Since US is sub-balanced, US consists of a balanced Union sequence US' on a subuniverse of $2^v > \frac{1}{2}n$ 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 TS' 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)-execution 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 show that E'' satisfies the conditions of Claim 5.1. Note that by Lemma 2.5 we have

$$a(i,n) \geq 8.12^i.i.(c+1)^{i-1}.(\,2a(i+1,n)+c+1). \tag{21}$$

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

$$\begin{aligned} &A(i, \lfloor \tfrac{1}{2}a(i,n') \rfloor - 1) \geq A(i, \lfloor \tfrac{1}{2}a(i,n) \rfloor - 2) \\ &\quad \geq A(i, 4.12^{i+1}.i.(c+1)^{i-1}.(\,2a(i+1,n)+c+1)-2) \geq A(i,(c+1)^i) \geq (c+1)^i \end{aligned}$$

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

Hence by Claim 5.1 the cost of E'' is at least

$$\frac{1}{12}.n'.a(1,n')$$

if $i=1$ and at least

$$\frac{1}{4.12^i.i.(c+1)^{i-1}}.n'.a(i,n') - (c+1).n'$$

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:*

> *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 $d.f.\alpha(f,n)$ if $\alpha(n,n) > \alpha(c,c) + 1$.*

Proof. Let n and c satisfy the constraints given above. Consider some sub-balanced Union sequence US on n elements. Let

$$i = max\{j | [f \leq \frac{n.a(j,n)}{j} \wedge 1 \leq j \leq \alpha(n,n) - 2] \vee j = 1\}. \tag{22}$$

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 $i \geq \frac{1}{3}.\alpha(f,n)$. 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 on-line, 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 2 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 $1 \leq i - 1 \leq \alpha(n,n) - 3$. 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 $f \cdot i \leq n.a(i,n)$. Hence the cost of E is at least $f.i$.

We show that $i \geq \frac{1}{3}.\alpha(f,n)$. We distinguish three cases.

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

Combining the above cases gives that $i \geq \frac{1}{3}.\alpha(f,n)$.

By combining the above results it follows that the cost is at least $\frac{1}{3}.f.\alpha(f,n)$. □

Theorem 5.3 implies that even if all Unions are known in advance, the worst case time bound is still $\Omega(f.\alpha(f,n))$ 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:*

> *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 $f \geq 0$ 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. 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 Split-Find problem the partition consists of one set only.) 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 $A \neq \emptyset \neq B$ 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.

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 define a Split sequence as a sequence of pairs $((A_k, B_k))_k$, where a pair (A_k, B_k) represents a Split operation Split(A_k, B_k). 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. (Note that we can also define Split Turns and sub-balanced Split sequences independent of Union Turns and Union sequences.) A UF(i, c)-execution of a Split sequence is defined similar 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:*

> *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:*

> *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 $d.(n+f.\alpha(f,n))$ 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 $d.(n + f.\alpha(f,n))$ 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 with 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 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.

8 Acknowledgements

I thank Jan van Leeuwen and Mark Overmars for useful comments.

References

[1] L. Banachowsky, A Complement to Tarjan's Result about the Lower Bound on the Complexity of the Set Union Problem, Inf. Process. Lett. 11 (1980), 59-65.

[2] N. Blum, On the Single-Operation Worst-Case Time Complexity of the Disjoint Set Union Problem, SIAM J. Comput. 15 (1986),1021-1024.

[3] P. v. Emde Boas, R. Kaas and E. Zijlstra, Design and Implementation of an Efficient Priority Queue, Math. Systems Theory 10 (1977), 99-127.

[4] M.L. Fredman and M.E. Saks, The Cell-Probe Complexity of Dynamic Data Structures, Proc. 21^{th} Ann. Symp. on the Theory of Comput. (STOC) (1989), 345-354.

[5] H.N Gabow, A Scaling Algorithm for Weighted Matching on General Graphs, Proc. 26^{th} Ann. Symp. on Found. of Comp. Sci. (FOCS) 1985, 90-100.

[6] H.N. Gabow and R.E. Tarjan, A Linear-Time Algorithm for a Special Case of Disjoint Set Union, J. Comput. Syst. Sci., no. 30, 1985, pp. 209-221.

[7] J.E. Hopcroft and J.D. Ullman, Set-Merging Algorithms, SIAM J. Comput. 2 (1973), 294-303.

[8] D.E. Knuth, The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, Massachusets, 1968.

[9] A.N. Kolmogorov, On the Notion of Algorithms, Uspekhi Mat. Nauk., 8 (1953), 175-176.

[10] J.A. La Poutré, New Techniques for the Union-Find Problem, Tech. Rep. RUU-CS-89-19, Utrecht University, 1989; an extended abstract will appear in: Proc. First Ann. ACM-SIAM Symp. on Discrete Algorithms (1990).

[11] J.A. La Poutré, A Fast and Optimal Algorithm for the Split-Find Problem on Pointer Machines, Tech. Rep. RUU-CS-89-20, Utrecht University, 1989.

[12] K. Mehlhorn, S. Näher and H. Alt, A Lower Bound for the Complexity of the Union-Split-Find Problem, SIAM J. Comput., vol. 17, no. 6 (1988), pp. 1093-1102.

[13] A. Schönhage, Storage Modification Machines, SIAM J. Comput. 9 (1980), 490-508.

[14] R.E. Tarjan, Efficiency of a Good but Not Linear Set Union Algorithm, J. ACM 22, No. 2, April 1975, pp 215-225.

[15] R.E. Tarjan, A Class of Algorithms which Require Nonlinear Time to Maintain Disjoint Sets, J. Comput. Syst. Sci. 18 (1979), 110-127.

[16] R.E. Tarjan and J. van Leeuwen, Worst-Case Analysis of Set Union Algorithms, J. ACM 31, No. 2, April 1984, pp. 245-281.