The Complexity of Finding Replicas Using Equality Tests[*][†]

Gudmund Skovbjerg Frandsen
Peter Bro Miltersen
Sven Skyum
Computer Science Department, Aarhus University

Abstract

We prove (for fixed k) that at least $\frac{1}{k-1}\binom{n}{2} - O(n)$ equality tests and no more than $\frac{2}{k}\binom{n}{2} + O(n)$ equality tests are needed in the worst case to determine whether a given set of n elements contains a subset of k identical elements. The upper bound is an improvement by a factor 2 compared to known results. We give tighter bounds for $k = 3$.

1 Introduction

When given n elements it is often possible to sort them according to bit string representation and hence decide which elements are identical in time $O(n \log n)$.

However, Campbell and McNeill [2] mention applications, where an equivalence relation is available, but it can not be extended to a total order on the bit representation. We mention one of their many examples. Consider a database of faces obtained by electronic scanning from different angles, where two images are equivalent if they represent the same person. If we want to verify that a set of images contains no duplicates, we may let a human being look at all possible image pairs, but it would seem to be of little help to sort the images according to bit representation.

[*]This research was supported by the ESPRIT II BRA Programme of the EC under contract # 7141 (ALCOM II) and by CCI-Europe.

[†]Accepted for presentation at MFCS'93.

This type of problem motivates the study of a computational model, where the only allowed operation is an equality test on pairs. We define two complexity measures:

Definition 1.1 (Existence) *For $2 \leq k \leq n$, let $E(k,n)$ denote the minimum number of equality tests that an algorithm must use in the worst case, to determine whether there exists k identical elements in a list of n elements.*

Definition 1.2 (Classification) *Similarly, define $C(k,n)$ to be the minimum number of equality tests that an algorithm must use in the worst case to find a representative for and the cardinality of each equivalence class with at least k members in a list of n elements.*

Clearly, $E(2,n) = C(2,n) = \binom{n}{2}$. This corresponds to the situation in the example above. If there are no duplicate images, that fact can be known to the algorithm only after comparison of all pairs.

However, the problem becomes nontrivial for $k > 2$. When checking the existence of triplicates, it is not necessary to compare all pairs of elements. We have managed to prove that $\frac{7}{12}\binom{n}{2} - O(n) \leq E(3,n) = C(3,n) \leq \frac{3}{5}\binom{n}{2} + 1$. This result determines the fraction of pairs that must be compared in the worst case up to $\frac{3}{5} - \frac{7}{12} = \frac{1}{60}$.

For general k the literature contains a single result, $C(k,n) \leq 2\lfloor\frac{n}{k}\rfloor n$ [2]. We have improved this result by almost a factor 2, and obtain $C(k,n) \leq (1 + \lfloor\frac{n}{k}\rfloor)n$. We have also found a general lower bound. Our results amount to $\frac{1}{k-1}\binom{n}{2} - \frac{n}{2} \leq E(k,n) \leq C(k,n) \leq \frac{2}{k}\binom{n}{2} + \frac{3n}{2}$, i.e. we determine the fraction of pairs that must be compared to within a factor less than 2. It is obvious that $E(k,n) \leq C(k,n)$, but we have not been able to prove that $E(k,n) = C(k,n)$ in general. Similarly, it is a trivial observation that $C(k,n)$ is nonincreasing in k, but we have no such monotonicity knowledge of $E(k,n)$.

Earlier special results have focused on determining a majority element, and the exact complexity $E(\lceil\frac{n+1}{2}\rceil, n) = C(\lceil\frac{n+1}{2}\rceil, n) = \lfloor\frac{3}{2}(n-1)\rfloor$ was found independently by several people [3, 4]. The interest in deciding the existence of a majority element was motivated by the design of fault tolerant computer systems where a majority of single processors must agree on the output [6]. The linear result is also interesting in a model where the elements are ordered [5].

In a restricted version of the majority problem, it is known in advance that there are precisely two distinct equivalence classes. It requires and suffices to make $n - B(n)$ equality tests to find a representative of the larger class, when n is odd ($B(n)$ denotes the number of ones in the binary representation of n) [7].

We have divided our results into two sections. In the first of these, we present our results for the special case $k = 3$, and in the last section we present our results for the general case.

2 The Complexity of Finding Triplicates

We start by proving that finding the cardinality of all equivalence classes with at least three elements is no harder that determining whether any such class exists. We continue by presenting and analysing a simple algorithm that determines the existence of a triplicate element.

The main result of this section is a lower bound for the same problem, where the proof uses an adversary argument combined with a graph theoretic view on the problem.

Theorem 2.1

$$E(3, n) = C(3, n)$$

Proof. Given n, let A be an algorithm that decides whether there exists 3 identical elements in an input consisting of n elements, using $E(3, n)$ equality tests in the worst case.

A can be regarded as a black box that once in a while poses an equality query. When fed the proper answers, it will eventually tell whether the input contains 3 identical elements or not. Since A never poses more than $E(3, n)$ queries it probably does some very efficient accounting inside the black box. Without knowledge of the nature of this internal accounting, we can nevertheless use A to construct an algorithm B that uses at most $E(3, n)$ equality queries to decide the cardinality of each equivalence class containing three or more elements and B will find a representative for each such class. Hence, the existence of B implies the theorem.

B starts the black box A. Each time A wants to know whether two elements x, y are identical then B makes the corresponding equality test, but B sometimes supplies a wrong answer to the black box A. B's strategy for deciding whether to supply A with the correct or the wrong answer is the following:

1. If the correct answer is "$\neq$" then this answer is passed on to A.

2. If the correct answer is "=" then B's action depends on the information A has received through earlier queries. If A when combining the correct answer "=" with old information may deduce the existence of 3 or more identical elements then B supplies the wrong answer "$\neq$" to A. Otherwise B gives A the correct answer "=".

When A stops, it must possess enough (possibly false) information to deduce that there are not three identical elements in the input. We claim that B at this point in time possesses enough (correct) information to determine all equivalence classes with three or more elements.

Let $X = \{x_1, \ldots, x_l\}$ be an equivalence class, $l \geq 3$. If A poses a query of the form $x_i?z$, where $z \notin X$, then A receives the correct answer "$\neq$" according to B's strategy, i.e. every equivalence class known to A is a subclass of a true equivalence class. Since $l \geq 3$, A must make at least one query of the form $x_i?x_j$. Without loss of generality, we may assume that the first such query is $x_1?x_2$. According to B's strategy, A receives the correct answer "=". Since A eventually concludes that $X' = \{x_1, x_2\}$ is a maximal equivalence class, then A must pose queries to reveal the relation between X' and any other equivalence class that it knows of. Since the class division known to A is a refinement of the true class division, B will through these queries obtain full knowledge of X. $\square$

Theorem 2.2

$$E(3,n) \leq \lceil \frac{3}{10} n(n-1) \rceil \leq \frac{3}{5} \binom{n}{2} + 1$$

Proof. We describe an algorithm that takes as input a list of elements and determines whether three of them are identical. The algorithm has two phases.

In the first phase, we take the input elements one by one and insert them in the smaller one of two sets U_i, $(i = 1, 2)$ such that we keep the sets equal sized, $|U_1| = |U_2| + \delta$, where $\delta \in \{0, 1\}$. We do not make any cross comparisons of U_1 and U_2, and hence we have no knowledge of $U_1 \cap U_2$. Each U_i is maintained as the disjoint union of two subsets: $U_i = S_i \cup D_i$, $S_i \cap D_i = \emptyset$, where S_i are the elements that have been seen only once, and D_i are the elements that we have seen twice. When inserting an element e into U_i, we first compare e to S_i. If $e \in S_i$ then we move e from S_i to

D_i. Otherwise we compare e to D_i. If $e \in D_i$ then we have seen e thrice and the algorithm halts, otherwise e was not in U_i and is inserted into S_i.

If the algorithm finishes the first phase without halting, we must compare U_1 to U_2 in the second phase. We need not make a complete pairwise comparison. If there is three identical elements then there is a representative in $D_1 \cup D_2$. Hence it suffices to compare S_1 to D_2, D_1 to D_2 and D_1 to S_2.

To prove the upper bound, we need to make a count of the number of equality tests that the algorithm uses in the worst case. Let $s_i = |S_i|$ and $u_i = |U_i|$. In phase one we may count the comparisons made when inserting into the two U_i's separately for each i. The worst sequence of events consists in first inserting u_i distinct elements into U_i followed by the insertion of $u_i - s_i$ duplicates. This uses a maximum number of comparisons

$$\binom{u_i}{2} + \binom{u_i+1}{2} - \binom{s_i+1}{2} = u_i^2 - \binom{s_i+1}{2}.$$

In phase two we need to make at most $u_1 u_2 - s_1 s_2$ comparisons.

We may express the total number of comparisons made in terms of $s = s_1 + s_2$ only, when using that $u_1 - u_2 = \delta$ ($\delta \in \{0, 1\}$) and $u_1 + u_2 = \frac{1}{2}(n+s)$:

$$\begin{aligned} & u_1 u_2 - s_1 s_2 + \Sigma_{i=1}^{2}[u_i^2 - \binom{s_i+1}{2}] \\ = \ & (u_1+u_2)^2 - u_1 u_2 - \tfrac{1}{2}s(s+1) \\ = \ & -\tfrac{5}{16}s^2 + (\tfrac{3}{8}n - \tfrac{1}{2})s + \tfrac{3}{16}n^2 + \tfrac{1}{4}\delta^2 \end{aligned}$$

The latter expression takes its maximum for $s = \frac{1}{5}(3n-4)$ and the integral part of the maximum expression value is at most $\lceil \frac{3}{10}n(n-1) \rceil$, which is an upper bound for $E(3,n)$. □

Theorem 2.3

$$E(3,n) \geq \frac{7}{24}n^2 - O(n) = \frac{7}{12}\binom{n}{2} - O(n)$$

Proof. We prove this lower bound by means of an adversary argument. The adversary maintains a complete graph on n vertices, corresponding to the n elements that the algorithm may compare pairwise in queries.

Initially, all edges in the graph are *black*. When answering a query $(x?y)$ the adversary paints the corresponding edge (x, y) *green* if the answer given is $x = y$ and otherwise the edge is painted *red*.

The adversary will always give answers that are consistent with no triple of elements being identical, i.e. the green edges will form a matching. On the other hand the information will for quite some time be consistent with the existence of three identical elements, i.e. the graph contains a 3-clique with no red edges for a long time.

The adversary will maintain a partition of the vertices in two classes, the green vertices G and the red vertices R. A vertex with an adjacent green edge is green and all other vertices are red.

In describing the adversary strategy, we need only consider the case, where the algorithm queries the relation $(x?y)$ for a black edge (x, y). If there would arise a clique of size $\frac{n}{3}$ with all vertices and edges red in case the edge (x, y) were painted red, then the adversary answers *equal* and otherwise it answers *distinct*.

The strategy preserves the following invariants:

1. Let C be a maximum size clique with all vertices and edges being red. Let $c = |C|$. It is always the case that $c < n/3$ and if $G \neq \emptyset$ then $c \geq n/3 - 2$. (To see the latter inequality, observe that immediately following the latest green-painting of an edge, $c \geq n/3 - 2$).

2. Let γ and ρ be the number of green and red *edges* respectively and let r be the number of red *vertices*. Then the number of distinct queries answered by the adversary is $\gamma + \rho$ and $\gamma = \frac{1}{2}(n - r)$.

We want to compute a lower bound for the number of queries made by an algorithm that verifies the nonexistence of three identical elements. Hence, it suffices to compute a lower bound for $\gamma + \rho$ when every 3-clique in the graph contains a red edge and the graph satisfies the invariants above. Clearly, $\gamma \leq n/2$ and since we want to prove a lower bound that hides linear terms under O-notation, we ignore γ and concentrate on ρ.

We shall find the number of red edges as a sum of four numbers, and for counting purposes we will regard a matched pair of green vertices as a supervertex.

1. Each pair of green supervertices are connected with a red edge, i.e. $\rho_1 = \frac{1}{2}\gamma(\gamma - 1)$.

2. Each red vertex is connected to each green supervertex with a red edge, i.e. $\rho_2 = r\gamma$.

3. We have not counted all red edges incident to green vertices. Two green super vertices may be connected by up to 4 red edges, and a red vertex and a green supervertex may be connected by 1 or 2 red edges. Our adversary strategy guarantees the existence of at least $n/3 - 2$ "extra" red edges for each green edge, i.e. $\rho_3 \geq (\frac{n}{3} - 2)\gamma$.

4. Finally, we count the red edges connecting red vertices. Note that the clique of red vertices contains only red and black edges, and each 3-clique in it must have at least one red edge. If x is a red vertex then let $N_b(x)$ ($N_r(x)$) denote all the red vertices that are connected to x by a black (red) edge. Observe that $N_b(x)$ must form a clique of red edges only, since we could otherwise find a 3-clique with black edges only. ¿From the invariant, we therefore know that $|N_b(x)| \leq c$, and consequently $|N_r(x)| \geq r - 1 - c$. If x lies in the clique C then we know in addition that $|N_r(x)| \geq c - 1$. We may combine this information and obtain for $x \in C$: $|N_r(x)| \geq \max(r - 1 - c, c - 1) \geq \frac{1}{2}[(r - 1 - c) + (c - 1)] = \frac{r}{2} - 1$. Adding up, we get $\rho_4 \geq \frac{1}{2}[c(\frac{r}{2} - 1) + (r - c)(r - c - 1)]$.

We may eliminate low order terms from the bound of ρ to obtain:

$$\begin{aligned} \rho &= \textstyle\sum_{i=1}^{4} \rho_i \\ &\geq \tfrac{1}{2}\gamma(\gamma - 1) + r\gamma + (\tfrac{n}{3} - 2)\gamma + \tfrac{1}{2}[c(\tfrac{r}{2} - 1) + (r - c)(r - c - 1)] \\ &= \tfrac{1}{2}\gamma^2 + r\gamma + \tfrac{n}{3}\gamma + \tfrac{1}{2}r^2 + \tfrac{1}{2}c^2 - \tfrac{3}{4}rc - O(n) \end{aligned}$$

We would like to eliminate c from this bound. From the invariant, we know that $\frac{n}{3} - 2 \leq c < \frac{n}{3}$, provided that $\gamma > 0$. If $\gamma = 0$, then $r = n$ and we find that $\rho \geq \frac{1}{2}c^2 - \frac{3n}{4}c + \frac{n^2}{2} - O(n)$. For $c < \frac{n}{3}$ the value of this expression is bounded from below by the value at $c = \frac{n}{3}$, which is $\frac{11}{36}n^2 - O(n)$. Hence, if $\gamma = 0$ then the algorithm has used at least as many equality tests as claimed in the statement of the theorem ($\frac{11}{36} > \frac{7}{24}$). In the following, we assume that $\gamma > 0$, which enable us to express the bound on ρ in terms of r only when using $c = \frac{n}{3} - O(1)$ and $\gamma = \frac{1}{2}(n - r)$:

$$\begin{aligned} \rho &\geq \tfrac{1}{8}(n - r)^2 + \tfrac{1}{2}r(n - r) + \tfrac{n}{6}(n - r) + \tfrac{1}{2}r^2 + \tfrac{1}{18}n^2 - \tfrac{1}{4}rn - O(n) \\ &= \tfrac{1}{8}r^2 - \tfrac{n}{6}r + \tfrac{25n^2}{72} - O(n) \end{aligned}$$

This expression takes its minimum value at $r = \frac{2n}{3}$, where the value is $\frac{7}{24}n^2 - O(n)$, which is a lower bound for $E(3, n)$.

The analysis of our adversary strategy is optimal, in the sense that the algorithm in the proof of theorem 2.2 uses at most $\frac{7}{24}n^2 + O(n)$ equality tests, when run against the adversary.

The essence of the adversary strategy is to answer "$\neq$", when there would otherwise arise a red clique of size $\frac{n}{3}$. The choice of $\frac{n}{3}$ gives the best lower bound among all possible fixed fractions of n. Hence, a significantly different adversary strategy is required for a possible improvement of the lower bound. □

3 The Complexity of Finding Replicas in General

We present a general lower bound with a simple proof that is based on Turan's theorem. The main result of this section is an efficient algorithm for the general case. The analysis of the algorithm is based on a rather intricate amortization argument.

Theorem 3.1

$$E(k,n) \geq \frac{n}{2}(\frac{n}{k-1} - 1) + k - 2 \geq \frac{1}{k-1}\binom{n}{2} - \frac{n}{2}$$

Proof. We rephrase our problem as a graph problem, and use an adversary argument combined with a result from extremal graph theory to prove the lower bound.

Observe that $E(k,n)$ is the minimum number of probes into the incidence matrix for a transitive graph G that will decide whether G contains a k-clique.

Our adversary strategy is the following: When an algorithm asks for an entry in the incidence matrix, we let the adversary answer that the edge is *not* present. So the algorithm will obtain information that is always consistent with the graph having no k-clique, but for quite some time the information will also be consistent with the graph having a k-clique. We need a lower bound on the number of probes for which the information remains nonconclusive. It is implied by the following result:

Fact 3.2 (Consequence of Turan's theorem [1, p. 293]) *Let $t_q(n) = \binom{n}{2} - \sum_{i=0}^{q-1}\binom{n_i}{2}$, where $n_i = \lfloor\frac{n+i}{q}\rfloor$. Every graph with n vertices and more than $t_{k-1}(n)$ edges contains a k-clique.*

The adversary strategy combined with the fact gives us the following lower bound:

$$E(k,n) \geq \binom{n}{2} - t_{k-1}(n).$$

When letting $r = n - (k-1)\lfloor \frac{n}{k-1} \rfloor$, we may compute

$$\begin{aligned} E(k,n) &\geq \Sigma_{i=0}^{k-2} \binom{n_i}{2} \\ &= (k-1-r)\binom{\frac{n-r}{k-1}}{2} + r\binom{\frac{n-r}{k-1}+1}{2} \\ &= \tfrac{(n-r)(n-(k-1-r))}{2(k-1)} \\ &\geq \tfrac{n}{2}(\tfrac{n}{k-1} - 1) \end{aligned}$$

The lower bound can be increased with $k-2$, if the adversary changes its strategy just before giving conclusive information. The adversary may then answer *equal* to at least $k-2$ queries without letting the algorithm resolve the situation. □

Theorem 3.3

$$C(k,n) \leq (\lfloor \frac{n}{k} \rfloor + 1)n \leq \frac{2}{k}\binom{n}{2} + \frac{3}{2}n$$

Proof. Let $l = \lfloor \frac{n}{k} \rfloor$, i.e. $\frac{n}{l+1} < k \leq \frac{n}{l}$. We shall describe an algorithm that receives at most ln "$\neq$"-answers to equality queries. This implies the theorem, since by proper accounting the algorithm will have complete information if it receives $n-1$ "="-answers to equality queries.

The algorithm works in two phases. In the first phase, the algorithm finds up to l elements satisfying that any equivalence class with at least k elements must have a representative among these l elements. In the second phase the abundance of the l candidates are checked.

In the first phase the algorithm inserts the elements one by one into a sequence of buckets $B_1, B_2, \ldots, B_m$, where m is initially 0 and increases according to need. All elements in a single bucket are identical and if the elements of two distinct buckets B_i, B_j are identical then their indices are at least $l+1$ apart, $|i-j| \geq l+1$. All buckets are nonempty, but only the first l buckets may contain two or more elements. Formally, if s_i is the number of elements in B_i, one of which is x_i, then we maintain the following invariants:

$$s_i \geq 1 \text{ for } 1 \leq i \leq \min(l,m)$$

$$s_i = 1 \text{ for } l < i \le m$$

$$1 \le |i - j| \le l \text{ implies that } x_i \ne x_j$$

When considering an unused element z we test whether it is identical to any of $x_1, x_2, \ldots, x_f$, where $f = \min(l, m)$. There are three cases:

1. If $z = x_i$, $i \le f$ then we add z to B_i (the invariants still hold).
2. If $m < l$ and z is distinct from all of $x_1, x_2, \ldots, x_m$ then we insert a new bucket B_{m+1} containing the single element z (invariants are preserved).
3. If $m \ge l$ and z is distinct from all of $x_1, x_2, \ldots, x_l$, the action is more complicated. This time we shift all buckets one up, and insert a new first bucket containing the single element z. At this point there may be more than one element in the updated B_{l+1}-bucket, which is not allowed according to the invariant. Exceeding elements from B_{l+1} are removed and inserted in a new first bucket following yet another shift. This continues until the invariant is satisfied. Formally, if $s_i \ge 2$ for all $1 \le i \le l$ then let $r = 0$ otherwise let r be maximum such that $1 \le r \le l$ and $s_r = 1$. If x'_i, s'_i, m' refers to the updated sequence of buckets then we have
   (a) $m' = m + l - r + 1$.
   (b) $x'_{i+l-r+1} = x_i$ and $s'_{i+l-r+1} = 1$ for $r \le i \le m$.
   (c) $x'_{i+l-r+1} = x_i$ and $s'_{i+l-r+1} = s_i$ for $1 \le i \le r - 1$.
   (d) $x'_{i+l-r} = z$ and $s'_{i+l-r} = 1$.
   (e) $x'_{i-r} = x_i$ and $s'_{i-r} = s_i - 1$, for $r + 1 \le i \le l$.

   One may check that this action preserves the invariants.

We have not quite finished the description of the first phase. When comparing z to $x_1, x_2, \ldots, x_f$, the comparison order is irrelevant, if no equality is found. However, if equality is found, it will be essential to the later complexity analysis that the following comparison order has been used: We compare z to an element of a larger bucket before comparing to an element of a smaller bucket, and among equal sized buckets, we compare z to an element from a bucket with small index before comparing z to an element from a bucket with large index. We stop comparing once equality is found.

Before describing the exact actions taken in the second phase, we make the following observation:

> Let $l \leq e \leq m$. Then the buckets $B_{e+1}, B_{e+2}, \ldots, B_m$ each contain exactly one element, and there are at least l buckets between two buckets with identical elements according to the invariant. This implies that if the listed buckets contain $q \geq 1$ copies of a specific element z, then $m-e \geq (q-1)(l+1)+1$, or equivalently $\phi(e,q) \geq 0$ where ϕ is defined as $\phi(e,q) = m - e - (q-1)(l+1) - 1$.
>
> Observe that $\phi(e+(l+1), q-1) = \phi(e,q)$ and $\phi(e+1,q) = \phi(e,q)-1$. We may now determine the number of copies of z (if it is at least q) in $B_{e+1}, \ldots, B_m$ as follows: While $\phi(e,q) \geq 0$ and $e < m$ compare z to x_{e+1}. If $z = x_{e+1}$ then $x_{e+2}, \ldots, x_{e+(l+1)}$ are all distinct from z and we increment e by $l+1$ and decrement q by one, which leaves $\phi(e,q)$ unchanged. If $z \neq x_{e+1}$ then we increment e by one and leave q unchanged, which decrements $\phi(e,q)$ by one.
>
> The sketched algorithm determines whether there are q copies of z in $B_{e+1}, \ldots, B_m$, and if so the algorithm finds the exact number of copies. It receives at most $\max(0, \phi(e,q)+1) = \max(0, m-e-(q-1)(l+1))$ "$\neq$"-answers.

This observation implies that we need only check the abundance of the elements in the first f buckets for $f = \min(l,m)$, since an element z distinct from all of $x_1, x_2, \ldots, x_l$ can only occur in k copies or more if $\phi(l,k) \geq 0$, i.e. if $n \geq m \geq k(l+1)$, which is contradictory to the definition of l.

We check the abundance of each of the f candidates $x_1, x_2, ..., x_f$ in turn. Let us consider the checking of an arbitrary one of these, $z = x_i$. By the invariant, we know that there are precisely s_i copies of z in the buckets $B_1, \ldots, B_{l+i}$. Hence, if $m \leq l+i$ then we are done and otherwise we check whether there are at least $k - s_i$ copies of z in the buckets $B_{l+i+1}, \ldots, B_m$ using the algorithm described in the observation.

We have now presented the algorithm and argued its correctness. We need still analyse its complexity. We argued earlier that it suffices to show that the algorithm never makes more than $l \cdot n$ comparisons that result in the answer "$\neq$". We compute the cost of the algorithm separately for the two phases:

Let C_1 be the number of "$\neq$"-answers that the algorithm receives when inserting the first v elements during the first phase, and let C_2 be the number of "$\neq$"-answers that the algorithm receives during the second phase, if it was executed directly following insertion of the first v elements in the first phase.

Let m, s_i denote the values taken by these variables following the insertion of the first v elements. Let $f = \min(l, m)$, let $t = \sum_{i=1}^{f} s_i$, let

$$F = \sum_{1 \le i < j \le f} \min(s_i, s_j)$$

and let

$$G = \#\{(i,j) | 1 \le i < j \le f \text{ and } s_i < s_j\}.$$

Then we have

Lemma 3.4

$$C_1 \le vl - tl - \tbinom{f}{2} + 2F + G$$

Lemma 3.5 *If $f < l$ then $C_2 = 0$ and if $f = l$ then*

$$C_2 \le tl + \tbinom{f}{2} - 2F - G$$

We note that the total cost of the algorithm is $C_1 + C_2 \le ln$ by the lemmas. Hence the theorem follows, when we have proved the lemmas:

Proof of lemma 3.4. We prove the inequality by induction on v. If $v = 1$ then $t = v = f = 1$ and the statement reduces to $C_1 \le 0$, which is true.

Assume that the inequality describes a correct upper bound for C_1. When inserting the $(v+1)$'th element z, the algorithm distinguishes three cases. We will argue that the invariant is preserved in each case. In the following we let unprimed variables refer to the situation before the insertion of z and we let primed variables refer to the situation after the insertion:

1. z is added to B_i. Let $J_1 = \{j | 1 \le j \le f \text{ and } s_i < s_j\}$ and let $J_2 = \{j | 1 \le j < i \text{ and } s_i = s_j\}$. By the comparison order, used by the algorithm, z is found distinct to other elements in precisely $\#J_1 + \#J_2$ comparisons, before the insertion into B_i. We note that $v' - t' = v - t$, $f' = f$, $F' = F + \#J_1$ and $G' \ge G + \#J_2 - \#J_1$. Hence, C_1' satisfies the stated inequality.

2. $m < l$ and z is found distinct from all of $x_1, \ldots, x_f$. We get f "$\neq$"-answers. $v' - t' = v - t = 0$, $f' = f + 1$, $F' = F + f$ and $G' = G$. Hence, C_1' satisfies the inequality in this case too.

3. $m \geq l$ and z is distinct from all of $x_1, \ldots, x_l$. We get l "$\neq$"-answers. Let r be defined as in the description of the algorithm. $v' = v + 1$, $t' = t - r$, $f' = f = l$ and $2F' + G' \geq 2F + G - (l - r)r$. Hence, in all three cases C_1' satisfies the inequality of the lemma.

Proof of lemma 3.5. The case of $f < l$ is trivial, hence we assume that $f = l$. By the observation in the description of the second phase, we know that

$$\begin{aligned}
C_2 &\leq \textstyle\sum_{i=1}^{f} \max(0, \phi(l+i, k-s_i) + 1) \\
&= \textstyle\sum_{i=1}^{f} \max(0, m - l - i - (k - s_i - 1)(l+1)) \\
&= \textstyle\sum_{i=1}^{f} \max(0, v - t - i - (k - s_i - 1)(l+1)) \\
&= \textstyle\sum_{i=1}^{f} \max(0, v - k(l+1) - t - i + (s_i + 1)(l+1)) \\
&\leq \textstyle\sum_{i=1}^{f} \max(0, -1 - t - i + (s_i + 1)(l+1)) \\
&= \textstyle\sum_{i=1}^{f} \max(0, s_i l + (l - i) - (t - s_i)) \\
&= \textstyle\sum_{i=1}^{f} \max(0, s_i l + (l - i) - \sum_{1 \leq j < i} s_j - \sum_{i < j \leq f} s_j) \\
&\leq \textstyle\sum_{i=1}^{f} \max(0, s_i l + (l - i) \\
&\quad - \textstyle\sum_{1 \leq j < i} \min(s_i, s_j) - \sum_{i < j \leq f} \min(s_i + 1, s_j)) \\
&= \textstyle\sum_{i=1}^{f} [s_i l + (l - i) \\
&\quad - \textstyle\sum_{1 \leq j < i} \min(s_i, s_j) - \sum_{i < j \leq f} \min(s_i + 1, s_j)] \\
&= tl + \tbinom{f}{2} + f(l - f) - 2 \textstyle\sum_{1 \leq i < j \leq f} \min(s_i, s_j) \\
&\quad - \#\{(i, j) | 1 \leq i < j \leq f \text{ and } s_i < s_j\}
\end{aligned}$$

The term $f(l - f)$ disappears, since we have assumed $l = f$. We have thus proved the lemma. □

References

[1] Bollobás, B., *Extremal Graph Theory.* Academic Press, 1978.

[2] Campbell, D. and McNeill, T., Finding a majority when sorting is not available. *The Computer Journal* **34** (1991) 186.

[3] Fischer, M. J. and Salzburg S. L., Solution to problem 81-5. *Journal of Algorithms* **3** (1982) 376–379.

[4] Matula, D. W., An Optimal Algorithm for the Majority Problem. Manuscript, Southern Methodist University, Texas, 1990.

[5] Misra, J. and Gries, D., Finding Repeated Elements. *Science of Computer Programming* **2** (1982) 143–152.

[6] Moore, J., Problem 81-5. *Journal of Algorithms* **2** (1981) 208–209.

[7] Saks, M. E. and Werman, M., On Computing Majority by Comparisons. *Combinatorica* **11** (1991) 383–387.