Selection Problems via m-ary Queries

Katia Guimaraes
University of Maryland

William Gasarch
University of Maryland

Jim Purtilo
University of Maryland

June 11, 1997

1 Introduction

It is easy to show that to find the maximum of n elements (from an infinite totally ordered set) $n-1$ comparisons are necessary and sufficient. More generally, to find the k-th largest of n elements, $n + (k-1)\log n + \Theta(1)$ comparisons are necessary and sufficient (originally due to [PY73], but see [FG79] for an elegant proof, and [Joh88] for more precise bounds).

Several natural questions arise: What is it about comparisons that is being used? Can these upper and lower bounds be extended to other types of queries? The lower bound cannot be extended to general boolean queries since the maximum of n elements can be found with $\lceil \log(n+1) \rceil$ such queries.

The key turns out to be the arity of the queries.

DEFINITION. An *m-ary boolean query* is an arbitrary statement involving m elements from the input. The answer is YES if the given statement is true, and NO otherwise. An *m-ary sort query* is a set of m elements. The answer is those m elements in order. If α is a type of query (e.g. m-ary

boolean) then an α *decision tree* is a decision tree where every internal node is labeled with a query of type α, and every possible answer to the query corresponds to a child of the node.

Moran, Snir and Manber [MSM85] studied m-ary decision trees. Although they did not mention selection problems, the following theorem is easily derivable from Theorem 3.5 in their paper.

Theorem 1 *If there exists a 2-ary boolean decision tree of height h that finds the k-th largest of n elements, then there exists a comparison decision tree of height h that finds the k-th largest of n elements.*

Corollary 2 *All known upper and lower bounds on the number of comparisons needed to solve a selection problem also hold for 2-ary boolean queries.*

Stated more succinctly, arbitrary 2-ary boolean queries are no better than comparions for selection problems. A generalization to m-ary queries, stated below, also follows from [MSM85].

DEFINITION. A query $Q(x_1, \ldots, x_n)$ is *order invariant* if it can be written as a boolean combination of predicates of the form $x_i < x_j$ and $x_i = x_j$.

Theorem 3 *If there exists an m-ary α decision tree of height h that finds the k-th largest of n elements then there exists an m-ary order invariant decision tree that finds the k-th largest of the n elements.*

In this paper we derive upper and lower bounds for the problem of determining the k-th largest of n elements using m-ary queries. By the above theorem, we need only consider m-ary order invariant queries. We consider boolean queries and sort queries.

NOTATION.

1. $V_k^{[m]}(n)$ is the number of m-ary boolean queries necessary to select the k-th largest element of an input set with n elements.

2. $V_k^{(m)}(n)$ is the number of m-ary sort queries necessary to select the k-th largest element of an input set with n elements.

3. $W_k^{[m]}(n)$ is the number of m-ary boolean queries necessary to select the k largest elements of an input set with n elements, and outputs them in increasing order.

4. $W_k^{(m)}(n)$ is the number of m-ary sort queries necessary to select the k largest elements of an input set with n elements, and output them in increasing order.

In Section 2 we study upper and lower bounds on the number of m-ary boolean queries necessary to select the k-th largest element of an input set. For the case of finding the maximum with $m = 3$ and $m = 4$ we have matching upper and lower bounds.

In Section 3 we derive exact bounds for $V_1^{(3)}(n)$ and $V_2^{(3)}(n)$, and provide reasonable bounds for $V_k^{(3)}(n)$ and $W_k^{(3)}(n)$. Since some of the proofs are somewhat detailed they are omitted; however they appear in [Guion].

2 Selection Problems Using Boolean Queries

In this Section we study boolean queries. Algorithms are represented by decision trees where each node represents a boolean query. A left branch from a node means that the answer to the query represented by that node is YES, whereas a right branch means a NO answer.

We first give an upper bound on the number of m-ary boolean queries necessary to select the k-th largest element of an input set. Then we present precise bounds on finding the maximum element with 3-ary and 4-ary boolean queries. For the case of 5-ary queries, we have a conjecture and some experimental evidence to support it. Throughout Section 2, whenever we use the term *query* we mean boolean query.

2.1 Upper Bounds

Lemma 4 *Let k and m be any natural numbers such that $k \leq m$. The k-th largest of m numbers can be found with $\lceil \log m \rceil$ m-ary queries.*

Proof: The question "Is the k-th largest element of $\{x_1, \ldots, x_m\}$ in the set $\{x_1, \ldots, x_{m/2}\}$?" is an m-ary query that cuts the number of elements that could be the maximum in half. Using this type of query and binary search, the k-th largest element can be located in $\lceil \log m \rceil$ m-ary queries. ☒

Theorem 5

$$V_1^{[m]}(n) \leq \begin{cases} \frac{2n}{m} + \lceil \log m \rceil - 1 & \textit{for } m \textit{ even} \\ \frac{2n}{m} + \lceil \log m \rceil - 2 & \textit{for } m \textit{ even and } m \textit{ divides } 2n \\ \frac{2n}{m-1} + \lceil \log m \rceil - 1 & \textit{for } m \textit{ odd.} \\ \frac{2n}{m-1} + \lceil \log m \rceil - 2 & \textit{for } m \textit{ odd and } m \textit{ divides } 2n \end{cases}$$

Proof: Assume m is even. Partition the input $(x_1, \ldots, x_n)$ into blocks $B_1, \ldots, B_{\lfloor \frac{2n}{m} \rfloor}, B_{\lfloor \frac{2n}{m} \rfloor + 1}$, where for $1 \leq i \leq \left\lfloor \frac{2n}{m} \right\rfloor$, $|B_i| = (m/2)$, and $0 \leq |B_{\lfloor \frac{2n}{m} \rfloor + 1}| < (m/2)$. Set up a tournament (see tournament sort in [Knu73]) where the blocks are the players, and the winner of a match between B_i and B_j is the answer to the m-ary query "Which of B_i or B_j contains the maximal element of $B_i \cup B_j$?" This tournament will determine the block B ($|B| \leq m/2$) that contains $\max(x_1, \ldots, x_n)$ and will use $\left\lfloor \frac{2n}{m} \right\rfloor$ m-ary queries. By Lemma 4 we

can find the maximal element of B using at most $\left\lceil \log(\frac{m}{2}) \right\rceil$ m-ary queries. Hence,

$$V_1^{[m]}(n) \leq \left\lfloor \frac{2n}{m} \right\rfloor + \left\lceil \log(\frac{m}{2}) \right\rceil \leq \frac{2n}{m} + \lceil \log m \rceil - 1.$$

If m divides $2n$ then only $\lfloor 2n/m \rfloor$ blocks are needed, so $V_1^{[m]}(n) \leq \frac{2n}{m} + \left\lceil \log(\frac{m}{2}) \right\rceil - 1 = \frac{2n}{m} + \lceil \log m \rceil - 2$.

The proof for m odd is similar, but uses blocks of size $(m-1)/2$. ☒

We will later show that for $m = 2, 3$, and 4 the upper bound cannot be improved. For bounds on $V_1^{[m]}(n)$ we roughly have:

m	2	3	4	5	6	7	...
$V_1^{[m]}(n)$	$n-1$	$n-1$	$n/2$	$n/2$	$n/3$	$n/3$	...

We conjecture that this table can be continued. In particular, we conjecture that arity $2k+1$ is no better than arity $2k$.

Theorem 6 *Let m be any number. If m is even then $V_k^{[m]}(n)$ is bounded above by*

$$\frac{2n}{m} + (k-1)\log(\left\lfloor \frac{2n}{m} \right\rfloor) + (k-1)\log^2(k) + \log(m) - (2k-1)\log(k) - 3k + O(1).$$

If m is odd then $V_k^{[m]}(n)$ is bounded above by

$$\frac{2n}{m-1} + (k-1)\log(\left\lfloor \frac{2n}{m} \right\rfloor) + (k-1)\log^2(k) + \log(m) - (2k-1)\log(k) - 3k + O(1).$$

Proof: Assume m is even. Assume both $\left\lfloor \frac{2n}{m} \right\rfloor$ and k are powers of 2. This case will differ from the general case by at most an additive constant.

Let $(x_1, \ldots, x_n)$ be the input. Throughout the algorithm we eliminate elements that cannot possibly be the kth largest element. The elements left are referred to as *candidates* .

Partition the input $(x_1, \ldots, x_n)$ into blocks or size $\leq m/2$, and build a tournament, as in the previous theorem. This uses $\frac{2n}{m} - 1$ m-ary queries.

Using $(k-1)\log\left\lfloor\frac{2n}{m}\right\rfloor$ additional queries, and the existing tournament, find the k "largest" blocks. The candidates are now the $km/2$ elements in those k blocks.

Repeat the above process on the $km/2$ candidates, using blocks of size $m/2^2$ and $m/2$-ary queries. Note that there are $2k$ blocks. The number of queries needed for this is

$$\text{number of blocks } + (k-1)\log(\text{ number of blocks })-1 = 2k+(k-1)\log(2k)-1;$$

the number of candidates left after this is $km/2^2$. If the process is repeated again, with blocks of size $m/2^3$ and $m/4$-ary queries, then the number of blocks is (again) $2k$, so the number of queries is (again) $2k+(k-1)\log(2k)-1$.

After $i \leq \log k - 1$ iterations of this process the total number of queries asked is

$$\left\lfloor\frac{2n}{m}\right\rfloor + (k-1)\log(\left\lfloor\frac{2n}{m}\right\rfloor) + i(2k + (k-1)\log(2k) - 1) - 1,$$

and the number of candidates is $(km/2^{i+1})$. Let $i = \log k - 1$. Now there are m candidates. By Lemma 4 the kth largest element can be found with $\log m$ m-ary queries. The total number of queries asked is

$$\left\lfloor\frac{2n}{m}\right\rfloor + (k-1)\log(\left\lfloor\frac{2n}{m}\right\rfloor) + (\log k - 1)[2k + (k-1)\log(2k) - 1] - 1 + \log m$$

which simplifies to

$$\left\lfloor\frac{2n}{m}\right\rfloor + (k-1)\log(\left\lfloor\frac{2n}{m}\right\rfloor) + (k-1)\log^2(k) + \log(m) - (2k-1)\log(k) - 3k + O(1).$$

For m odd assume both $\frac{2n}{m-1}$ and k are powers of two. For the initial block size use $\frac{2n}{m-1}$, and the proof is similar. ☒

2.2 Lower Bounds

We now precisely define the number of operations necessary to identify the maximum element of a set, using 3-ary and 4-ary queries.

NOTATION. An m-ary boolean query is represented by an ordered pair (Y, N) where Y is the set of all orderings of the m elements involved which make the query true, and N is the set of all other orderings.

Theorem 7 $V_1^{[3]}(n) = n - 1$.

Proof: The proof of Theorem 5 reveals that $V_1^{[3]}(n) \leq n - 1$.

We prove $V_1^{[3]}(n) \geq n - 1$ by showing

$$(\forall n \geq 2)[V_1^{[3]}(n+1) \leq n - 1 \Rightarrow V_1^{[3]}(n) \leq n - 2]$$

and using induction with base case $V_1^{[3]}(2) = 1$.

Assume $V_1^{[3]}(n+1) \leq n - 1$ via decision tree T. We construct a decision tree T' that shows $V_1^{[3]}(n) \leq n - 2$. There are three cases.

Case 1: The root of T is a 3-ary query (Y, N) where Y (N) contains the two orderings $x < y < z$ and $x < z < y$. Let T' be the tree formed by taking the left (right) subtree of T and removing x from any query by assuming x is the minimal element in the input tuple. Formally:
a) Eliminate any query of the form $x < p$ $(p < x)$ by assuming the answer is YES (NO).
b) Eliminate any 3-ary query (Y, N) where $x < p < q$ and $x < q < p$ are both in Y (N) by assuming the answer is YES (NO).
c) Let $Q = (Y, N)$ be a query where neither *a*) nor *b*) happens. Therefore Y contains the ordering $x < p < q$, and N contains the ordering $x < q < p$. Replace Q with $p < q$.
d) If x appears as the output of some leaf then it will appear in a part of the tree that gets eliminated by one of *a*), *b*) or *c*).

Case 2: The root of T is the query $x < y$. Similar to Case 1.

Case 3: The root of T is a 3-ary query (Y, N) such that Y (N) contains both $x < y < z$ and $z < x < y$. Let T' be the tree formed by taking the left

(right) subtree of T and removing x from any query by assuming that $x < y$ and no element is between the two. Formally,
a) Modify any query that contains x but not y by replacing all occurrences of x with y.
b) Eliminate any 3-ary query (Y, N) where $x < y < p$ and $p < x < y$ are both in Y (N) by assuming the answer is YES (NO).
c) Let $Q = (Y, N)$ be a comparison where neither a nor b happens. Therefore Y (N) contains the ordering $x < y < p$, and N (Y) contains the ordering $p < x < y$. Replace Q with $y < p$ $(p < y)$. (Recall that by convention, a YES answer means go down the YES side of the tree.)
d) If x appears as the output of some leaf then it will appear in a part of the tree that gets eliminated by one of a or b.

It is easy to see that in all cases, T' is of height $\leq n - 2$ and can be used to find the maximum of n elements.

We show that there are no more cases by attempting to construct a tuple that would not fit into any of the cases. Let Q be the query asked at the root. Q is not a single comparison, else Case 2 applies, so let $Q = (Y, N)$, a 3-ary query. Assume, without loss of generality, that $x < y < z \in L$. To avoid case 3, both $z < x < y$ and $y < z < x$ are in N. But having both of these in N is the case 3 with N instead of Y. $\boxtimes$

Theorem 8 $V_1^{[4]}(n) = \left\lceil \frac{n}{2} \right\rceil$.

Proof: The proof of Theorem 5 reveals that $V_1^{[4]}(n) \leq \left\lceil \frac{n}{2} \right\rceil$. We prove $V_1^{[4]}(n) \geq \left\lceil \frac{n}{2} \right\rceil$ by showing

$$(\forall n \geq 2)[V_1^{[4]}(n+2) \leq \left\lceil \frac{n}{2} \right\rceil \Rightarrow V_1^{[4]}(n) \leq \left\lceil \frac{n}{2} \right\rceil - 1]$$

and using induction with base case $V_1^{[4]}(2) = 1$.

Assume $V_1^{[4]}(n+2) \leq \left\lceil \frac{n}{2} \right\rceil - 1$ via decision tree T. We construct a decision tree T' that shows $V_1^{[4]}(n) \leq \left\lceil \frac{n}{2} \right\rceil - 2$. There are four cases.

Case 1: The root of T is a 4-ary query (Y, N) where Y (N) contains $x < y < z < w$ and $x < y < w < z$. Let T' be the tree formed by taking the left right subtree of T and removing x and y from any query by assuming that $x < y$ and that they are the bottom two elements.

Case 2: The root of T is either a 3-ary query or a comparison. Use the techniques of Theorem 7. The resulting tree finds the maximum of $n + 1$ elements.

Case 3: The root of T is a 4-ary query (Y, N) such that Y (N) contains both $x < y < z < w$ and $w < x < y < z$. Let T' be the tree formed by taking the left (right) subtree of T and removing x and y from any query by assuming that $x < y < z$ and no element is between x and y, or between y and z.

Case 4: The root of T is a 4-ary query (Y, N) such that Y (N) contains both $x < y < z < w$ and $y < x < w < z$. Let T' be the tree formed by taking the left (right) subtree of T and removing x and y from any query by assuming that x and y are the bottom two elements, and $x < y$ iff $z < w$.

It is easy to see that in all cases, T' is of height $\leq \left\lceil \frac{n}{2} \right\rceil - 1$ and can be used to find the maximum of n elements.

To show there are no more cases can be done easily by a case by case analysis, similar to that in Theorem 7. Details can be found in [Guion].

□

The case of 5-ary queries is open. However, we have a conjecture and some evidence to support it.

Conjecture 9 $V_1^{[5]}(n) = \left\lceil \frac{n}{2} \right\rceil$.

Evidence: The proof of Theorem 5 reveals that $V_1^{[5]}(n) \leq \left\lceil \frac{n}{2} \right\rceil$. We give

evidence for $V_1^{[5]}(n) \geq \left\lceil \frac{n}{2} \right\rceil$ by trying to show

$$(\forall n \geq 2)[V_1^{[5]}(n+2) \leq \left\lceil \frac{n}{2} \right\rceil \Rightarrow V_1^{[5]}(n) \leq \left\lceil \frac{n}{2} \right\rceil - 1]$$

and using induction with base case $V_1^5(2) = 1$.

Assume $V_1^{[5]}(n+2) \leq \left\lceil \frac{n}{2} \right\rceil - 1$ via decision tree T. We construct a decision tree T' that shows $V_1^{[5]}(n) \leq \left\lceil \frac{n}{2} \right\rceil - 2$. There are four cases.

Case 1: The root of T is a 5-ary query (Y, N) where Y (N) contains all orderings that have $x < y$ as the bottom two elements. Let T' be the tree formed by taking the left (right) subtree of T and removing x and y from any query by assuming that $x < y$ and they are the bottom two elements.

Case 2: The root of T is either a 4-ary query, a 3-ary query, or a comparison, then use the techniques of Theorem 8.

Case 3: The root of T is a 5-ary query (Y, N) such that Y (N) contains all orderings that have $x < y < z$ and nothing in between. Let T' be the tree formed by taking the left (right) subtree of T and removing x and y from any query by assuming that $x < y < z$ and no element is between the two.

Case 4: The root of T is a 5-ary query (Y, N) such that Y (N) contains all orderings where both $x < w$ (and nothing is between them), and $y < u$ (and nothing is between them). Let T' be the tree formed by taking the left (right) subtree of T and removing x and y from any query by assuming that $x < w$, $y < u$, and in both cases there is nothing in between the two.

It is easy to see that in all cases, T' is of height $\leq \left\lceil \frac{n}{2} \right\rceil - 2$ and can be used to find the maximum of $n - 1$ elements.

We now examine our conjecture that all decision trees fall into one of the four cases enumerated above. To date, a direct method to demonstrate this conjecture has eluded us, and we have therefore focused on mechanical checks to suggest evidence. Consider all queries involving $\{x, y, z, v, w\}$, and

label these strings $S = s_1, ..., s_{120}$. A partition (Y, N) is any subset L of S, with its compliment R. Let *TEST* refer to our mechanical check that a given (Y, N) meets one of the cases enumerated in our conjecture.

To mechanically generate all partitions on five-ary queries would be taxing, amounting to some 1.3×10^{37} cases. This can be reduced (slightly) due to symmetry in the cases: one only need examine partitions of size sixty, since any partition (Y, N) with L of size larger than sixty can be relabelled to reverse the roles of L and R. Still this is an unwieldy number of cases to check exhaustively.

In order to gain insight on the conjecture, we therefore considered partitions (Y, N) where each was much *smaller* than sixty. If we were to discover that even the smaller query sets still fell within the necessary seven cases of the conjecture, then certainly the supersets all having size sixty would also satisfy the conjecture.

A collection of test programs were generated by the Minion system [Pur89]. Some of these programs were in C and some were in Macsyma, a computer algebra system. The Polylith software interconnection system [PJar] allowed interoperation between these programs, and also allowed different test cases to be smoothly distributed across the available computing resources. Generation of partitions having size 8 or less proved unfruitful — most partitions required additional strings in order to satisfy one of the *TEST* criteria. Next, partitions having size 10 appeared to be a tractible objective. Unfortunately, while most query sets satisfied *TEST*, several cases still arose where the set required additional strings in order to satisfy the *TEST*. In all cases checked manually we were able to show how any extension to the partition needed to bring it up to the correct size would also satisfy *TEST*; nonetheless, the manual check prohibited our evaluating all such cases.

Finally, we considered generation of partitions having size 15 and greater. It is computationally infeasible for us to apply *TEST* to all such partitions, so we instead elected to *TEST* a selected subset of all cases to see if we could

quickly discover anomalous cases. Our experiences with previous cases revealed that the regular methods for generating partitions resulted in 'runs' where the presence of one key subset would always result in the same *TEST* case being satisfied. We therefore generated partitions based on a permutation set that could be randomized, so the cases we could afford to check would not all necessarily fall to the same *TEST*. All partitions of size 15 that we checked in this manner satisfied *TEST* without requiring manual adaptation or checks. While we still are short of having exhausted all cases, these experiences are suggestive of our conjecture's validity.

3 Selection Problems Using 3-sort Queries

We now study bounds on selection problems using 3-sort queries. Throughout section 3, whenever we use the term *query* we mean 3-sort query. We assume that the input contains n distinct elements, and we represent a selection algorithm which uses 3-sort queries by a 3-sort tree.

In this section we precisely determine $V_1^{(3)}(n)$ and $V_2^{(3)}(n)$, and we derive upper and lower bounds for $V_k^{(3)}(n)$ and $W_k^{(3)}(n)$.

3.1 Finding the Maximum of n Elements

The bounds for selecting the maximum element can be derived easily.

Theorem 10 $V_1^{(3)}(n) = \lceil (n-1)/2 \rceil$.

Proof: To show the upper bound, we present the following algorithm:

Algorithm 1.

1. If $n = 1$, then return the one element; if $n = 2$, then compute 3-sort(a_1, a_2, a_2), and return the maximum; if $n = 3$, then compute 3-sort(a_1, a_2, a_3), and return the maximum;

2. If $n > 3$, then let $i \equiv n \pmod 3$:

   (a) Partition the input into subsets of 3 elements, leaving i elements out;

   (b) Apply 3-sort queries to the subsets of the input;

   (c) Recursively apply the algorithm to the set of maximum elements of subsets and the i elements left out in step 2.a.

End of Algorithm 1.

This algorithm clearly identifies the maximum element of a set, and the number of 3-sort queries that it requires is given by the following recursive equation:

$$T(n) = \begin{cases} n/3 + T(n/3), & \text{if } n \bmod 3 = 0 \ ; \\ (n-1)/3 + T((n-1)/3+1), & \text{if } n \bmod 3 = 1 \ ; \\ (n-2)/3 + T((n-2)/3+2), & \text{if } n \bmod 3 = 2 \ . \end{cases}$$

It can be easily shown by induction that: $T(n) = \lceil (n-1)/2 \rceil$. Since the algorithm finds the maximum element in $T(n) = \lceil (n-1)/2 \rceil$ queries, we have shown that $V_1^{(3)}(n) \leq \lceil (n-1)/2 \rceil$.

As for the lower bound, recall that in each operation at most two elements are discarded as potential maximum element. Since $n-1$ elements must be discarded, we have that $V_1^{(3)}(n) \geq \lceil (n-1)/2 \rceil$.

Thus, $V_1^{(3)}(n) = \lceil (n-1)/2 \rceil$. ☒

NOTE. Let S(n) be the number of 3-sort stages to find maximum of n elements using Algorithm 1. S(n) $\leq \lceil \log_3(n) \rceil + 1$.

3.2 Finding the Second Largest of n Elements

DEFINITION. An element a is said to *be defeated by* another element b if $(a < b < c)$, or $(a < c < b)$, or $(c < a < b)$ is the outcome of some query. Reciprocally, an element is said to be *undefeated* if it was the largest in any query in which it was involved so far.

Theorem 11 $V_2^{(3)}(n) \leq \lceil (n + \lceil \log_3(n) \rceil)/2 \rceil$.

Proof: We use Algorithm 2, which is an extension of Algorithm 1, from Theorem 10.

Algorithm 2: Apply Algorithm 1 to select the maximum, but in the process keep track of which elements were second to each of the undefeated elements. After selecting the maximum element, m, find the maximum among the elements which were defeated only by m.

Since the number of stages that Algorithm 1 uses is $\leq \lceil \log_3(n) \rceil + 1$, there are at most $\lceil \log_3(n) \rceil + 1$ candidates for second largest left after the maximum has been identified. Applying the upper bound for $V_1^{(3)}(n)$, we have:

$$V_2^{(3)}(n) \leq \begin{cases} n/2 + \lceil \log_3(n) \rceil/2, & \text{if } n \text{ even and } \lceil \log_3(n) \rceil \text{ even;} \\ n/2 + (\lceil \log_3(n) \rceil + 1)/2, & \text{if } n \text{ even and } \lceil \log_3(n) \rceil \text{ odd;} \\ (n-1)/2 + \lceil \log_3(n) \rceil/2, & \text{if } n \text{ odd and } \lceil \log_3(n) \rceil \text{ even;} \\ (n-1)/2 + (\lceil \log_3(n) \rceil + 1)/2, & \text{if } n \text{ odd and } \lceil \log_3(n) \rceil \text{ odd.} \end{cases}$$

Studying each case, it can be verified that $V_2^{(3)}(n) \leq \lceil (n + \lceil \log_3(n) \rceil)/2 \rceil$. ☒

We now prove a very close lower bound. Our techniques are similar to those of Pratt and Yao [PY73].

DEFINITIONS. Let m be the maximum and s be the second largest element in the set. Let Q_1, Q_2, ..., Q_q be a sequence of 3-sort queries such that if asked in that order one can determine s at the end.
A 3-sort query Q_i is said to *be critical for* an element x if:

i) Q_i involves x and y, where $x < y \leq s$, or $s = y < x = m$; and

ii) Q_i is the first such query in the sequence.

A 3-sort query Q is said to be *partly useless* if:

i) some element y is involved twice in Q, e.g. (y, y, x), or

ii) some element y which was defeated previously by an element $z \leq s$, is defeated again in the outcome of Q by an element $z' \leq s$.

The partial order defined by the outcomes of the queries until a given point of execution is called the *Current Partial Order (CPO)* .

If x is an element of the input set,

$$weight(x) = |\{y\text{: } (y \leq x) \text{ is in the CPO}\}|.$$

Or in words, $weight(x)$ is the number of elements of the input known to be $\leq x$. Notice that $weight(x) \geq 1$, since $x \leq x$. We use the idea of *weight* to establish the lower bound in this subsection, and to refine both upper and lower bounds in the next subsection.

Lemma 12 *For all $x \neq s$, there must be a query that is critical for x in the sequence Q_1, Q_2, ..., Q_q .*

Proof: It is easy to see that in the resulting partial order each element x of the set has to be comparable with s. Otherwise, total orders having $x > s$ or $x < s$ are consistent with the outcome of the algorithm, and one cannot guarantee that s is the second largest element in the set. ☒

Corollary 13 *A 3-sort tree which determines the second largest element also identifies the maximum.*

Proof: Follows trivially from the fact that there is a query which is critical for m in the sequence. ☒

Lemma 14 *A 3-sort query cannot be critical for more than two elements. If a 3-sort query involves m but not s, then it can be critical for at most one element. If a 3-sort query involves both m and s then it can be critical for two elements; however, any later query involving m and s is critical for at most one element.*

Proof: Let $(a < b < c)$ be the outcome of a 3-sort query. If Q_i does not involve m, then $a < b < c \leq s$, and by definition Q_i can only be critical for a or b, but not c. If Q_i involves m but not s, then $a < b < s < c = m$, and again by definition, Q_i can only be critical for a. Finally, if Q_i involves both m and s, then $a < b = s < c = m$. In that case, Q_i may be critical for a or c, but only on the first time that m and s are involved in the same query. After that, once again by definition, a query involving m and s can only be critical for a. ☒

An Adversary Strategy.

We now introduce an adversary strategy to decide the outcome of a 3-sort query. This strategy is a modification of the Basic Strategy used in [PY73]. We also call it the *Basic Strategy* .

Basic Strategy: Given a 3-sort query $\mathrm{Q} = (x, y, z)$, assume without loss of generality that $weight(x) \leq weight(y) \leq weight(z)$. Let the outcome of Q be $x < y < z$.

Notice that the outcome cannot lead to a contradiction because once an ordering has been established between two elements, the weight of the larger will always subsume the weight of the smaller element.

Lemma 15 *If the Basic Strategy is used, then there is a path from the root to a leaf of a 3-sort tree to determine the second largest element, with at least* $\lceil \log_3(n) \rceil$ *queries involving the maximum element.*

Proof: Let S be a 3-sort tree to determine second largest. ¿From Corollary 13, S also determines the maximum element, m. Using induction on k, one can show that, following the Basic Strategy, after an element x is involved in k 3-sort queries, $weight(x) \leq 3^k$. Hence, if $weight(m) = n$, then m must have been involved in $\lceil \log_3(n) \rceil$ queries. ☒

Theorem 16 $V_2^{(3)}(n) \geq \lceil (n + \lceil \log_3(n) \rceil)/2 \rceil - 1$.

Proof: By Lemma 15, any algorithm to determine the second largest element will have at least $\lceil \log_3(n) \rceil$ 3-sort queries involving the maximum element. By Lemma 14, those queries can be critical for at most $\lceil \log_3(n) \rceil + 1$ elements. By Lemma remaining $n - \lceil \log_3(n) \rceil - 2$ elements. Again by Lemma 14, $\lceil (n - \lceil \log_3(n) \rceil - 2)/2 \rceil$ other queries are necessary. Thus, $V_2^{(3)}(n) \geq \lceil (n + \lceil \log_3(n) \rceil - 2)/2 \rceil \geq \lceil (n + \lceil \log_3(n) \rceil)/2 \rceil - 1$. ☒

Theorem 17 $\lceil (n + \lceil \log_3(n) \rceil)/2 \rceil - 1 \leq V_2^{(3)}(n) \leq \lceil (n + \lceil \log_3(n) \rceil)/2 \rceil$.

Proof: Follows from Theorems 11 and 16. ☒

Theorem 17 can be improved to obtain matching upper and lower bounds. We omit the somewhat detailed proof; however it is in [Guion].

Theorem 18

$$V_2^{(3)}(n) = \begin{cases} \lceil (n + \lceil \log_3(n) \rceil - 1)/2 \rceil, & \text{if } n \text{ is even and } n > 2 \times 3^{\lceil \log_3(n) \rceil - 1}; \\ \lceil (n + \lceil \log_3(n) \rceil - 2)/2 \rceil, & \text{if } n \text{ is odd or } n \leq 2 \times 3^{\lceil \log_3(n) \rceil - 1}. \end{cases}$$

3.3 Bounds for $W_k^{(3)}(n)$ and $V_k^{(3)}(n)$

We can extend Algorithm 2 in the same fashion in which it was extended from Algorithm 1 to obtain an algorithm that selects the third largest element using $O(\log_3 \log_3 n)$ more queries than the lower bound.

As k gets larger, this scheme of algorithm becomes less and less efficient with respect to the lower bound.

For the following two theorems, techniques similar to tree selection [Knu73] can be used to obtain algorithms that establish upper bounds; and the lower bound is proven in a manner similar to Theorem 16, details can be found in [Guion].

Theorem 19

$$W_k^{(3)}(n) \geq \lceil (n-k+\lceil \log_3(n \times (n-1) \times \cdots \times (n-k+2))\rceil)/2\rceil$$

and

$$W_k^{(3)}(n) \leq \lceil (n-1)/2\rceil + (k-1) \times \lceil \log_3 n\rceil .$$

Less precisely, there exists a constant β, $1 \leq \beta \leq 2$, *such that*

$$\frac{n-1}{2} + (\frac{k-1}{2}\log_3 n) - \beta(k) \leq W_k^{(3)}(n) \leq \frac{n-1}{2} + ((k-1)\log_3 n).$$

Theorem 20

$$\lceil (n-k+(k-1)\times\lceil \log_3(n-k+2)\rceil)/2\rceil \leq V_k^{(3)}(n) \leq \lceil (n-k)/2\rceil + (k-1)\times\lceil \log_3(n-k+2)\rceil$$

3.4 Conclusion

We conjecture that for boolean queries, the pattern stated after Theorem 5 persists beyond the $n = 4$ case.

We believe that the bounds that we presented for $W_k^{(3)}(n)$ and for $V_k^{(3)}(n)$ can be improved. Future work in this area includes finding lower bounds for $V_k^{[m]}(n)$ and $W_k^{[m]}(n)$, and better bounds for $W_k^{(3)}(n)$ and for $V_k^{(3)}(n)$.

References

[FG79] Fussenegger and Gabow. A counting approach to lower bounds for selection problems. *JACM*, 26(2):227–238, April 1979.

[Guion] Katia Guimaraes. *Topics in Concrete Complexity Theory.* PhD thesis, Dept. of Computer Science - U. of Maryland at College Park, in preparation.

[Joh88] John W. John. A new lower bound for the set-partitioning problem. *SIAM J. Computing*, 17(4):640–647, Aug 1988.

[Knu73] Donald E. Knuth. *The Art of Computer Programming*, volume 3 - Sorting and Searching. Addison Wesley, 1973.

[MSM85] S. Moran, M. Snir, and U. Manber. Applications of Ramsey's theorem to decision tree complexity. *JACM*, 32:938–949, 1985.

[PJar] Jim Purtilo and Pankaj Jalote. An enviroment for prototyping distributed applications. *Computer Languages*, To appear.

[Pur89] Jim Purtilo. An envirorment to organize mathematical problem solving. In *Proc. of the 1989 International Symposium on Symbolic and Algebraic Computation*, pages 147–154, 1989.

[PY73] Vaughan Pratt and Foong Frances Yao. On lower bounds for computing the i-th largest element. In *Proc. of the 14th IEEE Symp. on Switching and Automata Theory*, pages 70–81, 1973.