A Deterministic Algorithm for Selection

We present a deterministic algorithm SELECT for selection which just like QUICKSELECT, chooses a pivot in each iteration, splits the remaining elements depending on whether they are bigger or smaller than the pivot and then recurses on the subproblem containing the rth order statistic.

The expected number of comparisions made by Quickselect is linear because using a uniformily random choice of pivot ensures that it is likely that the pivot will split the elements into two subsets which are not that far from equal. Our new algorithm uses a pivot selection subroutine which guarantees that the pivot chosen splits the elements into two pieces each with almost three-tenths of the elements.

The pivot-choosing subroutine is reasonably complex and uses SELECT as a subroutine. It is set out on page 220 of CLRS and proceeds as follows:

1. Partition A into $\lceil \frac{n}{5} \rceil$ groups, all of which except possibly one have size 5.

2. Find the median of each group of size at most 5 by first sorting the group.

3. Use SELECT to find the median (if n is odd) or lower median of the set of medians of the groups. Call this element p.

The rest of the algorithm proceeds as in QUICKSELECT.

4. Using p as a pivot and $n-1$ comparisions partition $A - \{p\}$ into the set Low of elements smaller than p and the set $High$ of elements larger than p.

5. If $r = |Low| + 1$ return p, if $r < |Low| + 1$ recursively apply SELECT to obtain the rth order statistic of Low. If $r > |Low| + 1$ then recursively apply SELECT to find the $(r - |Low| - 1)th$ order statistic of High.

As discussed in CLSR and shown in class, in Step 5, we recurse on a problem with at most $\frac{7n}{10} + 6$ elements. In Step 3, we apply Select to an instance of size at most $\frac{n}{5} + 1$. We can carry out the remaining steps in $O(n)$ time. Now, provided n is sufficiently large the sum of the sizes of the two problems we recurse on is at most $\frac{19n}{20}$. So, intuitively we have lost total size $\frac{n}{20}$ in our subproblems using $O(n)$ work in Steps 1,2, and 4. Since this is a constant amount of work for each element we get rid of, intuitively the whole algorithm should take linear time. Indeed as shown in CLRS and class, this intuition is correct and SELECT runs in $O(n)$ time.

Lower Bounds on The Number of Comparisions Performed by a Sorting or Selection Algorithms

In this section, we discuss lower bounds on the number of comparisions needed to sort or select the ith order statistic of a set $A = \{a_1, ..., a_n\}$ of n distinct elements. We will assume that the only way that we can obtain information about the relative order of two of these elements is to compare them. We can think of an oracle to which we give an i and j which returns whether $a_i < a_j$ or $a_j < a_i$. Dependent on the results of the comparisions made so far, we either perform a further comparision or terminate with the desired output.

We are thus considering the decision tree model discussed in Section 8.1 of CLRS. A decision tree is a rooted tree all of whose non-leaf nodes have exactly two children. Each interior node of the paths from the root to the leaves is labelled by an ordered pair of distinct integers (i, j) with $1 \leq i, j \leq n$. Each leaf is labelled by a solution which will be the output if we arrive at it.

The paths of the tree from the root to the leaves represent the possible sequences of comparisions made and responses obtained by the algorithm on its inputs. For every input we follow a unique such path to arrive at a leaf. This path is defined as follows. Each input begins its journey at the root. Having arrived at an interior node labelled (i, j), inputs for which $a_i < a_j$ go to the left child of this node while inputs for which $a_j > a_i$ go the right child.

For the algorithm to be correct, for every possible input the answer labelling the leaf at which we arrive when given this input must be the correct solution for this input. The worst case number of comparisions made is the longest such path, i.e. the height of the tree.

As an example, we describe for every n, a tree T_n of height $n - 1$ which computes the maximum of n elements $a_1, ..., a_n$. To each leaf l of the tree, we associate a label $max(l)$ such that $a_{max(l)}$ is the maximum of the elements assuming we arrive at the leaf.

T_1 has a single node whch is both a root and a leaf. It is labelled with 1 since a_1 must be the maximum. T_2 has a single interior node the root, which is labelled $(1, 2)$. Its left child is a leaf labelled 2 and its right child is a leaf labelled 1.

T_n is obtained from T_{n-1} as follows. We add a right and left child underneath each leaf of T_{n-1}, which are leaves of the new tree. So the leaves of

T_{n-1} are now interior nodes of the new tree. Any such interior node which was labelled i in T_{n-1} is relabelled (i, n). Its left child is labelled n and its right child is labelled i. We can prove by induction on n that T_n correctly computes the maximum and has height $n - 1$.

Information Theory Lower Bound For Sorting

We can obtain a lower bound on the height of a decision tree for a specfic problem by obtaining a lower bound on the number of leaves it must have. To prove such a lower bound, we exploit the fact that a rooted binary tree of height l has at most 2^l leaves(this can be easily proven by induction by consdering the two subtrees rooted at the children of the root). Such a bound is referred to as an *information theory* lower bound as it depends on the amount of information that the output provides.

To illustrate this approach, we consider sorting. There are $n!$ different permutations of $\{a_1, ..., a_n\}$. Each of these is the unique correct answer to at least one input for our sorting problem. Thus, any decision tree for sorting must have at least $n!$ leaves. So any decision tree for soritng has height at least $log(n!)$. But $log(n!)$ is $nlog\ n + o(n)$.

0.1 Adversarial Lower Bounds For Sorting and Selection

We can also obtain a lower bound for solving a sorting or selection problem by specifying a long path of the decision tree. Of course, given the whole tree it is easy to find the longest path. We take a slightly different approach and consider an adversary who will find a long path from the root to a leaf one node at a time without knowing the full tree. To do so, he specifies at each node of the path in turn, beginning at the root, the answer to the comparision at that node which leads to the next node on the path. The only information he has when doing so is the part of the tree he has seen so far. To prove a lower bound of l on the length of the path the adversary constructs, it is enough to give an adversarial strategy which ensures that that the length of the path constructed is l, no matter what the comparisions on the path are. These bounds are called *adversarial*.

The information theory lower bound on sorting can also be thought of as an adversarial lower bound. The adversary's strategy focuses on the number

of permutations of A which are consistent with the answers to the comparisions made before reaching a node. At a leaf of the decision tree, the number of such consistent permutations is at most one, as otherwise the algorithm may give an incorrect answer. At the root, all n! permutations are consistent. The adversary chooses the answer to the comparision at the node so as to maximize the number of permutations consistent with it and the answers to all the previous comparisions. If these two numbers are equal it chooses $a_i < a_j$. Since each permutation consistent with the previous answers is consistent with exactly one of the answers to the comparision at the current node, the number of consistent comparisions at most halves at each node. So, the longest path has length at least $\lceil log(n!) \rceil$.

Consider for example $n = 3$, so there are $3! = 6$ permutations of A. If the comparision at the root is (i, j), there are three permutations consistent with $a_i < a_j$ and three consistent with $a_j < a_i$ depending on whether the third element of A is first, second, or third in the permutation. So, the adversary selects the answer $a_i < a_j$ and so the path grows to include the left child.

This child cannot be a leaf as there are not 1 but 3 comparisions consistent with the answer to the single comparision made so far. If the child makes the comparision (i, k) for $k \notin \{i, j\}$ then if we answer $a_i < a_k$ there is one permutation[1] which remains consistent, while if we answer $a_i < a_k$ there are two permutations[2] which remain consistent. So the adversary choose $a_i < a_k$ and the path will also need a node labelled (j, k) to complete the sort.

If the corresponding child again makes the comparision (i, j) then the same answer $a_i < a_j$ must be given and the number of consistent comparisions does not decrease. Hence in this case we need at least one more node on the path (in fact two) before we arrive at the leaf. We obtain the correct lower bound, 3, on the number of comparision snecessary to sort a set of size three.

Adversarial arguments can be much more sophisticated. To illustrate, we consider median finding. For simpliciy we assume n is odd. I.e $n = 2k+1$ for some integer k. We will give a lower bond of $3k$ on the number of comparisions needed to find the median.

Our adversarial strategy proceeds as follows.We are going to label all of the elements which have been involved in comparisions from:

$W = \{w_1, ..., w_k\}, L = \{l_1, ..., l_k\}$ and *median*.

Each element of A gets a distinct label. The nodes not yet involved in a

[1] a_k, a_i, a_j

[2] a_i, a_j, a_k and a_i, a_k, a_j

comparision are all unlabelled.

Initially, the elements are all unlabelled. We will ensure that as the algorithm proceeds the following holds

(*) for some n_W and n_L with $1 \leq n_W, n_L \leq k$, the algorithm will have used the labels $w_1, ...w_{n_W}$ and $l_1, .., l_{n_L}$. The last element to be involved in a comparision is labelled *median*.

When comparing nodes, we will insist that (i) any element labelled w_i beats any element with a label in $L \cup median \cup \{w_j | j < i\}$ and loses to every element with a label in $\{w_j | j > i\}$. We insist further that (ii) any element labelled l_i loses to any element with a label in $W \cup median \cup \{l_j | j > i\}$ and beats every element with a label in $\{l_j | j < i\}$. This means that an element labelled *median* loses to elements with a label from W and beats elements with a label from L. When labelling an uncompared node we choose its label before performing the comparision and then respect (i) and (ii).

If we can ensure that our labelling respects (*), since our answers respect (i) and (ii), at any time, for any labelling obtained by assigning each uncompared element a distinct unused label, we see that there is an input whose median is labelled *median* which will have arrived at the current node of the decision tree. It is the input where looking at the elements in order of increasing size we see the labels in the following order:

$$l_1, l_2, ..., l_k, median, w_1, ..., w_k$$

Thus, at a leaf, there is an input which is consistent with the answers made so far for which the median is the element labelled *median*. Thus the output for this leaf must be the element labelled *median*. Hence, at a leaf, every input permutation consistent with the answers made so far must have this element as a median. If there is an i such that the element labelled l_i has not lost a comparision to any element with a label in $L \cup \{median\}$, then the permutations where the labels appear in the order

$$l_1, .., l_{i-1}, l_{i+1}, ..., l_k, median, l_i, w_1, ..., w_k$$

would also be consistent with the answers given so far, which is a contradiction. In the same vein, if there is an i such that the element labelled w_i has not won a comparision against any element with a label in $W \cup \{median\}$, then the permutations where the labels appear in the order

$$l_1, ..., l_k, w_i, median, w_1, ..., w_{i-1}, w_{i+1}, ..., w_k$$

would also be consistent with the answers given so far, which is a contradiction,

So, at a leaf every every element with a label in L has lost a comparision to an element with a label in $L \cup \{median\}$ and there have been at least k comparisions involving two elements with labels in $L \cup \{median\}$. In the same vein, every every element with a label in W has won a comparision against an element with a label in $W \cup \{median\}$ and there have been at least k comparisions involving two elements with labels in $W \cup \{median\}$.

In particular this implies there are no uncompared elements in $L \cup W$. We claim that we can chose a labelling procedure ensuring that (*) remains true and such that, at all times, there have been at least $max(n_W, n_L)$ comparisions involving one element with a label in L and the other in W. Since at a leaf there are no uncompared elements in $L \cup W$, we must have $n_W = n_L = k$ and hence if our claim is true, there are at least k such comparisions and $3k$ comparision in total.

It remains to prove our claim by describing and analyzing our labelling procedure. Intially we have $n_W = n_L = 0$.
For comparisions which compare an unlabelled element to an element with a label in L we proceed as follows:

If $n_W < k$ then $n_W := n_W + 1$ and the uncompared element is labelled w_{n_W}

else if $n_L < k$ then $n_L := n_L + 1$ and the uncompared element is labelled l_{n_L}

else the uncompared element is labelled *median*.

For comparisions which compare an unlabelled element to an element with a label in W we proceed as follows:

If $n_L < k$ then $n_L := n_L + 1$ and the uncompared element is labelled l_{n_L}

else if $n_W < k$ then $n_W := n_W + 1$ and the uncompared element is labelled w_{n_W}

else the uncompared element is labelled *median*.

For comparisions which compare two unlabelled element we proceed as follows:

If $n_L < k$ and $n_W < k$ then $n_w := n_W + 1, n_L := n_L + 1$ one uncompared element is labelled l_{n_L} and the other is labelled w_{n_W}

elseif $n_L < k - 1$ then $n_L := n_L + 2$ one uncompared element is labelled l_{n_L} and the other is labelled $l_{n_L - 1}$

elseif $n_W < k-1$ then $n_W := n_W + 2$ one uncompared element is labelled w_{n_W} and the other is labelled w_{n_W-1}

else there are only two unused labels, we label the elements in the comparision with these labels and set $n_W = n_L = k$.

We claimed that throughout the process, the number of comparisions made between elements in W and elements in L is at least $min(n_W, n_L)$.. To see this note first thay we increase each of n_W and n_L by at most one in each iteration. Note furthermore, that when we increase one of these numbers, either the comparision made is between an element of W and an element of L or the other number had value k and hence $max(n_W, n_L)$ did not increase. The claim follows.