Algorithms for Selection

Ref: Chapter 9 of text.

Definition: Given a set $A$ of $n$ elements, the $i^{th}$ order statistic of $A$ is the $i^{th}$ smallest element of $A$.

Examples:
- First order statistic: Minimum value.
- $n^{th}$ order statistic: Maximum value.
- Medians: Occur at positions $\lfloor (n + 1)/2 \rfloor$ (the lower median) and $\lceil (n + 1)/2 \rceil$ (the upper median). The two positions coincide when $n$ is odd.

Note: We use “median” to mean the lower median.

Selection problem:

Input: A set of $n$ distinct numbers and an integer $i$, $1 \leq i \leq n$.

Output: the $i^{th}$ order statistic of $A$.

Easy solution: $O(n \log n)$ time (sorting).

Better: $O(n)$ algorithm (this lecture).

Some special cases:

(a) Finding the minimum.
- Can be done using $(n - 1)$ comparisons (trivial algorithm).
- Can this be improved? No. $(n - 1)$ comparisons are necessary.

Proof: Think of the algorithm as conducting a “knock out” tournament among the $n$ numbers. Each comparison $x_i : x_j$ is a match and the winner is the smaller of $x_i$ and $x_j$. 
Observation: Every number, except the ultimate winner, must have lost at least one match. So, the number of matches = number of comparisons $\geq n - 1$.

(b) Finding the minimum and the maximum simultaneously.

- Can be done using $\lceil 3n/2 \rceil$ comparisons (divide and conquer).
- At least $\lceil 3n/2 \rceil - 2$ comparisons are necessary (proof omitted).

A simple algorithm for selection:

- Use Partition (of Quicksort) to partition the given array $A[p .. r]$ into the low side (i.e., the subarray $A[p .. q - 1]$), the pivot (i.e., $A[q]$) and the high side (i.e., the subarray $A[q + 1 .. r]$).
- Let $k = q - p + 1$ (i.e., $k$ is the number of elements on the low side plus the pivot).

- If $i = k$, return $A[q]$. If $i < k$, then recursively find the $i^{th}$ smallest value on the low side; otherwise, recursively find the $(i - k)^{th}$ smallest value on the high side.

Note: After Partition, Quicksort works on both the low and high sides. The selection algorithm works on only one side.

Worst-case performance: The recurrence is

$$W(n) \leq W(n - 1) + cn$$

with $W(1) = c_1$. The solution is

$$W(n) = O(n^2).$$

Exercise: Construct an example to show that the running time of the simple algorithm is $\Omega(n^2)$.

A randomized selection algorithm:

- Outline same as the simple selection algorithm, except that instead of Partition, use Randomized-Partition.

Pseudocode: Handout 10.1.
Expected Running time: $O(n)$. (Proof omitted).

**Selection in linear time (worst-case):**

Algorithm Select: Basic idea

- Recursive partitioning as before.
- The split is guaranteed to be “good”.

Outline of Select: Handout 10.1.

**Analysis of Select:**

**Lemma 1:** After the partitioning in Step 4,
(a) Low side has at least $\frac{3n}{10} - 6$ elements.
(b) High side has at least $\frac{3n}{10} - 6$ elements.

**Proof:** To be discussed in class.

**Lemma 2:** After the partitioning in Step 4, the larger side of the partition has at most $\frac{7n}{10} + 6$ elements.

**Proof:** Immediate from Lemma 1.

**Recurrence:** (to be proven in class)

$$T(n) \leq c_3 n + T(\lceil n/5 \rceil) + T(7n/10 + 6)$$

for some constant $c_3 > 0$ and $n > 140$. Also, $T(n) = \Theta(1)$ for $n \leq 140$.

**Solution:** $T(n) = O(n)$ (to be discussed in class).