Medians and Order Statistics

Jeff Hicks

Edited by Chuck Cusack

These notes are based on chapter 16 of [1] and lectures from CSCE423/823, Spring 2001.

1 Basic Definitions

Definition 1 *The* **ith order statistic** *of a set of n elements is the ith smallest element in the set.*

Definition 2 *A* **minimum** *of a set is the first order statistic ($i = 1$).*

Definition 3 **maximum** *of a set is the nth order statistic ($i = n$).*

Definition 4 **median** *of a set is the "halfway point". When n is odd, the median occurs at $i = (n + 1)/2$. When n is even, the median occurs at $i = n/2$ and $n/2 + 1$.*

Problem 1 *Given a set A of n distinct numbers and a number i, with $1 \leq i \leq n$, the* **selection problem** *asks to find an element $x \in A$ such that x that is larger than exactly $i - 1$ other elements of A. In other words, find the ith order statistic.*

The selection problem can easily be solved in $O(nlgn)$ time using heapsort or merge sort and then indexing the ith element in the output array. We will see, however, that the problem can be solved in time $O(n)$ in the worst case.

2 Minimum and Maximum

The following algorithm returns the minimum element in a set A.

```
MINIMUM (A)
1   min= A[1]
2   for(i=2  to length[A])
3      if(min >  A[i])
4         min=A[ i]
5   return  min
```

It is easy to modify this algorithm to compute the maximum in A. It is not hard to see that if $|A| = n$, then this algorithm requires $n - 1$ comparisons, which is the best one can hope for. To find the minimum and maximum, we can use these two algorithms, requiring $2(n - 1)$ steps. However, we can do better than this. If we compare each pair of elements, then compare the smaller to the min and the larger to the max, we can find both the minumum and the maximum with only $\lceil 3n/2 \rceil - 2$ comparisons.

3 Selection in expected linear time

In this section we present a divide-and-conquer algorithm for the selection problem. The algorithms is based on the `RandomPartition` algorithm used with `Quicksort`.

```
Random-Select(A,l,r,k)
1   m = RandomPartition(A,l,r)
2   if (k ==  m - l +1) then return A[m]
3   else if (k < m - l +1) then return Random-Select(A,l,m-1,k)
4   else return Random-Select(A,m+1,r,k-(m - l + 1))
```

The algorithm partitions about a random element, and continues the search on whichever side of the pivot k is on.

We will analyze the average-case running time of `Random-Select`.

Lemma 1 *Let $T(n)$ be the average case complexity of* `Random-Select`. *Then*

$$T(n) = \frac{2}{n} \sum_{i=\frac{n}{2}+1}^{n-1} T(i) + O(n)$$

Proof: If we assume that the larger of the partitions is always taken, we have that

$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} max(T(i), T(n-1-i)) + O(n)$$

We shall assume that $T(n)$ is monotomically increasing (which is reasonable), so that $T(n-1-i)$ is larger when $0 \leq \frac{n}{2}$, and $T(i)$ is larger otherwise. Thus, we have

$$\begin{aligned} T(n) &= \frac{1}{n} \sum_{i=0}^{\lceil n/2 \rceil} T(n-1-i) + \frac{1}{n} \sum_{i=\lceil \frac{n}{2} \rceil + 1}^{n-1} T(i) + O(n) \\ &= \frac{1}{n} \sum_{i=\frac{n}{2}+1}^{n-1} T(i) + \frac{1}{n} \sum_{i=\frac{n}{2}+1}^{n-1} T(i) + O(n) \\ &= \frac{2}{n} \sum_{i=\frac{n}{2}+1}^{n-1} T(i) + O(n) \end{aligned}$$

□

Theorem 2 *The average case complexity of* `Random-Select` *is $O(n)$.*

Proof: We need to show that $T(n) \leq cn$ for some $c > 0$ and all $n \geq 1$. Notice that when $n = 1$, we need $T(1) \leq cn = c$. We can choose c sufficiently large so that this holds. Now assume that $T(i) \leq ci$ for $i < n$. Then

$$\begin{aligned}
T(n) &= \frac{2}{n}\sum_{i=\frac{n}{2}+1}^{n-1} T(i) + O(n) \\
&\leq \frac{2}{n}\sum_{i=\frac{n}{2}}^{n-1} ci + O(n) \\
&\leq \frac{2c}{n}\sum_{i=\frac{n}{2}}^{n-1} i + O(n) \\
&\leq \frac{2c}{n}\left(\sum_{i=1}^{n-1} i - \sum_{i=1}^{\frac{n}{2}-1}\right) + O(n) \\
&\leq \frac{2c}{n}\left((n-1)\frac{n}{2} - (\frac{n}{2}-1)\frac{n/2}{2}\right) + O(n) \\
&\leq \frac{2c}{n}(\frac{3}{8}n^2 - \frac{n}{4}) + O(n) \\
&\leq \frac{3}{4}cn - \frac{c}{2} + O(n) \\
&\leq cn
\end{aligned}$$

assuming we can pick c so that

$$O(n) \leq \frac{1}{4}cn + \frac{c}{2},$$

which we can. The proof of this is left to the reader. Thus, we have $T(n) \leq cn$ for all $n \geq 1$, so $T(n) = O(n)$. □

4 Selection in worst-case linear time

Although the algorithm in the last section has a good average case, like quicksort, it has a worst-case complexity of $O(n \log n)$ We now examine a selection algorithm whose running time is $O(n)$ in the worst case. Like `Random-Select`, `Select` will find the desired element by recursively partitioning the input array. However, it will choose the pivot element so that the partition will be even, guaranting a linear runtime.

The algorithm is as follows.

`Select`(A, l, r, k)

1. Split array A into $\lceil n/5 \rceil$ groups of size 5 (except possibly last group).
2. Insertion sort each group.
3. Let B=array of medians of each group from A.
4. M=`Select`$(B, 1, n/5, (n/5)/2)$
5. `PartitionAbout`$(A, l, r, pivot)$
6. if $(k < p)$ return `Select`$(A, l, p-1, k)$
   else if $(k > p)$ return `Select`$(A, p+1, r, k)$
   else return p

To be continued...

References

[1] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw Hill, 1990.