Introduction to Algorithms

Chapter 9 : Medians and Order Statistics

XiangYang Li and Haisheng Tan[1]

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2024

Outline of Topics

9.1 Minimum and Maximum

9.2 Selection in Expected Linear Time
Overview
Analysis

9.3 Selection in Worst-case Linear Time
Overview
Analysis

Selection Problem

- This chapter addresses the problem of selecting the i-th order statistic from a set of n distinct numbers. We formally specify the selection problem as follows:
  Input: A set A of n (distinct) numbers and an integer i, with $1 \leq i \leq n$.
  Output: The element $x \in A$ that is larger than exactly $i-1$ other elements of A.

Minimum and Maximum

- To determine the minimum of a set of n elements, a lower bound of comparisons is $n-1$.
- The following procedure selects the minimum from the array A, where $A.length = n$.

```
MINIMUM(A)
min = A[1]
for i = 2 to A.length do
    if min > A[i] then
        min = A[i]
return min
```

Simultaneous Minimum and Maximum

- In some applications, we must find both the minimum and the maximum of a set of n elements.
- A simple solution: find the minimum and maximum independently, using $n-1$ comparisons for each, for a total of $2n-2$ comparisons.
- In fact, we can find both the minimum and the maximum using at most $3\lfloor n/2 \rfloor$ comparisons.

Simultaneous Minimum and Maximum

MAX-MIN(A)

if $A[1] > A[2]$ then $\min = A[2], \max = A[1]$
    else $\min = A[1], \max = A[2]$
for $i = 2$ to $\lfloor n/2 \rfloor$ do
    if $A[2i-1] > A[2i]$
        then if $A[2i] < \min$ then $\min = A[2i]$
            if $A[2i-1] > \max$ then $\max = A[2i-1]$
        else if $A[2i-1] < \min$ then $\min = A[2i-1]$
            if $A[2i] > \max$ then $\max = A[2i]$
if $n \neq 2\lfloor n/2 \rfloor$ then if $A[n] < \min$ then $\min = A[n]$
    if $A[n] > \max$ then $\max = A[n]$
return $(\min, \max)$

Simultaneous Minimum and Maximum

- Total number of comparisons:
  If n is odd, then we perform $3\lfloor n/2 \rfloor$ comparisons. If n is even, we perform 1 initial comparison followed by $3(n-2)/2$ comparisons, for a total of $3n/2-2$. Thus, in either case, the total number of comparisons is at most $3\lfloor n/2 \rfloor$.

Selection in Expected Linear Time

- A divide-and-conquer algorithm for the selection problem: RANDOMIZED-SELECT.
- The idea is to partition the input array recursively (as in quick-sort).
- The difference is that quick-sort recursively processes both sides of the partition, but RANDOMIZED-SELECT only works on one side of the partition.
- Quick-sort has an expected running time of $\Theta(n \log n)$, but the expected time of RANDOMIZED-SELECT is $\Theta(n)$.

RANDOMIZED-SELECT

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r then
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1
if i == k then
    return A[q] // the pivot value is the answer
if i < k then
    return RANDOMIZED-SELECT(A, p, q − 1, i)
else
    return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

RANDOMIZED-SELECT - Analysis

- The worst-case running time for RANDOMIZED-SELECT is $\Theta(n^2)$, even to find the minimum, because we could be extremely unlucky and always partition around the largest remaining element, and partitioning takes $\Theta(n)$ time.
- The expected running time for RANDOMIZED-SELECT is $\Theta(n)$.

RANDOMIZED-SELECT - Analysis

- The time required by RANDOMIZED-SELECT on an input array $A[p \dots r]$ of n elements is denoted by $T(n)$.
- We define indicator random variables X_k where $X_k = I\{$ the subarray $A[p \dots q]$ has exactly k elements $\}$. So we have $E[X_k] = \frac{1}{n}$, X_k has the value 1 for exactly one value of k, and it is 0 for all other k. When $X_k = 1$, two subarrays on which we might recurse have sizes $k-1$ and $n-k$

RANDOMIZED-SELECT - Analysis

$$\begin{aligned} T(n) &\le \sum_{k=1}^{n} X_k(T(\max(k-1,n-k))+O(n)) \\ &= \sum_{k=1}^{n} X_k T(\max(k-1,n-k))+O(n) \end{aligned}$$

$$\begin{aligned} E[T(n)] &\le E[\sum_{k=1}^{n} X_k T(\max(k-1,n-k))+O(n)] \\ &= \sum_{k=1}^{n} E[X_k T(\max(k-1,n-k))]+O(n) \\ &= \sum_{k=1}^{n} E[X_k]E[T(\max(k-1,n-k))]+O(n) \\ &= \sum_{k=1}^{n} \frac{1}{n}E[T(\max(k-1,n-k))]+O(n) \end{aligned}$$

RANDOMIZED-SELECT - Analysis

$$\max(k-1, n-k) = \begin{cases} k-1 \text{ if } k > \lceil n/2 \rceil, \\ n-k \text{ if } k \leq \lceil n/2 \rceil \end{cases}$$

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E(T(k)) + O(n)$$

- Assume that $T(n) \leq cn$ for some constant c that satisfies the initial conditions of the recurrence. Pick a constant a such that the function described by the $O(n)$ term above (which describes the non-recursive component of the running time of the algorithm) is bounded from above by an for all $n > 0$.

RANDOMIZED-SELECT - Analysis

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} ck + an \\
&= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2\rfloor -1} k\right) + an \\
&= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2\rfloor - 1)\lfloor n/2\rfloor}{2}\right) + an \\
&\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\le \frac{3cn}{4} + \frac{c}{2} + an = cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)
\end{aligned}
$$

RANDOMIZED-SELECT - Analysis

- For sufficiently large n, we have

$$n(\frac{c}{4} - a) \geq \frac{c}{2}$$

- As long as we choose the constant c so that $c/4 - a > 0$, i.e., $c > 4a$, we can divide both sides by $c/4 - a$, giving

$$n \geq \frac{c/2}{c/4 - a} = \frac{2c}{c - 4a}$$

- If we assume that $T(n) = O(1)$ for $n < \frac{2c}{c-4a}$, we have $T(n) = O(n)$.
- So any order statistic, and in particular the median, can be determined on average in linear time.

Selection in Worst-case Linear Time

- We now examine a selection algorithm whose running time is O(n) in the worst case. Like RANDOMIZED-SELECT, the algorithm SELECT finds the desired element by recursively partitioning the input array.
- The SELECT algorithm determines the i th smallest of an input array of n > 1 distinct elements by executing the following steps. (If n $= 1$, then SELECT merely returns its only input value as the i th smallest.)

Selection in Worst-case Linear Time

- 1. Divide the n elements of the input array into $\lfloor n/5 \rfloor$ groups of 5 elements each and at most one group made up of the remaining $n \mod 5$ elements.
- 2. Find the median of each of the $\lfloor n/5 \rfloor$ groups.
- 3. Use SELECT recursively to find the median x of the $\lfloor n/5 \rfloor$ medians found in step 2.
- 4. Partition the input array around the median-of-medians x using the modified version of PARTITION. So that x is the kth smallest element and there are $n-k$ elements on the high side and $k-1$ elements on the low side.
- 5. If $i = k$, then return x. Otherwise, use SELECT recursively to find the i th smallest element on the low side if $i < k$, or the $(i-k)$ th smallest element on the high side if $i > k$.

Selection in Worst-case Linear Time - Analysis

- To analyze the running time of SELECT, we first determine a lower bound on the number of elements that are greater than the partitioning element x.
- At least half of the $\lceil n/5 \rceil$ groups contribute 3 elements that are greater than x, except for the one group that has fewer than 5 elements if 5 does not divide n exactly, and the one group containing x itself. So the number of elements greater than x is at least

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

- So in the worst case, SELECT is called recursively on at most $\frac{7n}{10} + 6$ elements in step 5.

Selection in Worst-case Linear Time - Analysis

- Steps 1, 2, and 4 take $O(n)$ time.
- Step 3 takes time $T(\lceil n/5 \rceil)$, and step 5 takes time at most $T(7n/10+6)$, assuming that T is monotonically increasing
- Assume that any input of 140 or fewer elements requires $O(1)$ time.
- So we have the recurrence

$$T(n) \le \begin{cases} \Theta(1) & \text{if } n \le 140, \\ T(\lceil n/5 \rceil) + T(7n/10+6) + O(n) & \text{if } n > 140. \end{cases}$$

Selection in Worst-case Linear Time - Analysis

- Assuming that $\mathrm{T(n)} \leq \mathrm{cn}$ for some suitably large constant c and all $\mathrm{n} \leq 140$.
- Pick a constant a such that the function described by the $\mathrm{O(n)}$ term above is bounded above by an for all $\mathrm{n} > 0$.
- So we have

$$\begin{aligned}
\mathrm{T(n)} &\leq \mathrm{c}\lceil \mathrm{n}/5 \rceil + \mathrm{c}(7\mathrm{n}/10 + 6) + \mathrm{an} \\
&\leq \mathrm{cn}/5 + \mathrm{c} + 7\mathrm{cn}/10 + 6\mathrm{c} + \mathrm{an} \\
&= 9\mathrm{cn}/10 + 7\mathrm{c} + \mathrm{an} \\
&= \mathrm{cn} + (-\mathrm{cn}/10 + 7\mathrm{c} + \mathrm{an})
\end{aligned}$$

Selection in Worst-case Linear Time - Analysis

- Thus T(n) is at most cn if($-cn/10 + 7c + an \leq 0$)

  $$c \geq 10a(n/(n-70)) \text{when } n > 70$$

  Because $n \geq 140$ $\quad n/(n-70) \leq 2$
  So choosing $c \geq 20a$ will satisfy inequality.
- The worst-case running time of SELECT is therefore linear.
- The algorithm is still correct if each group has r elements where r is odd and is not less than 5.

Selection in Worst-case Linear Time - Analysis

- Sorting requires $\Omega(n \log n)$ time in the comparison model, even on average, and the linear-time sorting algorithms in Chapter 8 make assumptions about the input.
- But the linear-time selection algorithms in this chapter do not require any assumptions about the input.
- The running time is linear because these algorithms do not sort.