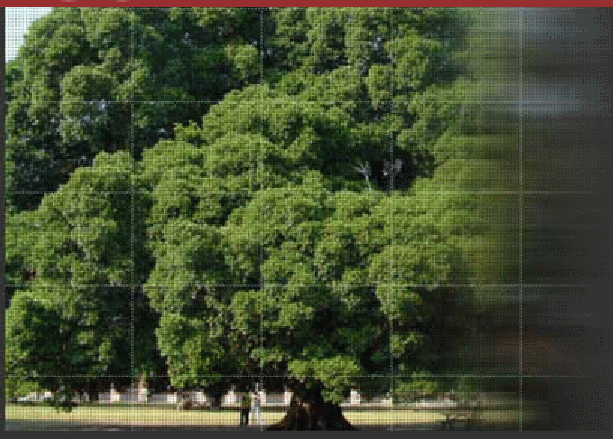
National Cheng Kung University



Lecture Notes for Chapter 9: Medians and Order Statistics

Sun-Yuan Hsieh

謝孫源 教授

成功大學資訊工程學系

Chapter 9 overview

- ***ith order statistic*** is the ith smallest element of a set of n elements
- The ***minimum*** is the first order statistic ($i = 1$).
- The ***maximum*** is the nth order statistic ($i = n$).
- A ***median*** is the "halfway point" of the set.
- When n is odd, the median is unique, at $i = (n + 1) / 2$.
- When n is even, there are two medians:
  - The ***lower median***, at $i = n / 2$, and
  - The ***upper median***, at $i = n / 2 + 1$.
  - We mean lower median when we use the phrase "the median."

The selection problem

Input: A set A of n distinct numbers and a number i, with $1 \leq i \leq n$.

Output: The element $x \in A$ that is larger than exactly $i - 1$ other elements in A. In other words, the ith smallest element of A.

The selection problem can be solved in $O(n\lg n)$ time.

- Sort the numbers using an $O(n\lg n)$-time algorithm, such as heap sort or merge sort.
- Then return the ith element in the sorted array.

Minimum and Maximum

We can easily obtain an upper bound of $n - 1$ comparisons for finding the minimum of a set of n elements.

- Examine each element in turn and keep track of the smallest one.
- This is the best we can do, because each element, except the minimum, must be compared to a smaller element at least once.

Minimum and Maximum

The following pseudocode finds the minimum element in array $A[1\ldots n]$:

MINIMUM (A, n)
$min \leftarrow A[1]$
for $i \leftarrow 2$ **to** n
    do if $min > A[i]$
        then $min \leftarrow A[i]$
return min

The maximum can be found in exactly the same way by replacing the > with < in the above algorithm.

Simultaneous minimum and maximum

- A simple algorithm to find the minimum and maximum: $n - 1$ comparisons for the minimum and $n - 1$ comparisons for maximum ⇒ total $2n - 2$ comparisons ⇒ $\Theta(n)$ time.

- In fact, at most $3 \lfloor n / 2 \rfloor$ comparisons are needed to find both the minimum and maximum !

Simultaneous minimum and maximum (cont.)

- Compare the elements of a pair to each other.
- Then compare the larger element to the maximum so far, and compare the smaller element to the minimum so far.

This leads to only 3 comparisons for every 2 elements.

Setting up the initial values for the min and max depends on whether *n* is odd or even.

- If *n* is even, compare the first two elements and assign the larger to max and the smaller to min. Then process the rest of the elements in pairs.
- If *n* is odd, set both min and max to the first element. Then process the rest of the elements in pairs.

Analysis of the total number of comparisons

- If n is even, we do 1 initial comparison and then $3(n-2)/2$ more comparisons.

  # of comparisons $$= \frac{3(n-2)}{2} + 1 = \frac{3n-6}{2} + 1 = \frac{3n}{2} - 3 + 1 = \frac{3n}{2} - 2.$$

- If n is odd, we do $3(n-1)/2 = 3\lfloor n/2 \rfloor$ comparisons.

In either case, the maximum number of comparisons is $\leq 3\lfloor n/2 \rfloor$.

Selection in expected linear time

RANDOMIZED-SELECT(A, p, r, i)

if $p = r$

  then return $A[p]$

$q \leftarrow$ RANDOMIZED-PARTITION (A, p, r)

$k \leftarrow q - p + 1$

if $i = k$ ► pivot value is answer

  then return $A[q]$

elseif $i < k$

  then return RANDOMIZED-SELECT(A, p, $q - 1$, i)

else return RANDOMIZED-SELECT(A, $q + 1$, r, $i - k$)

Selection in expected linear time

After the call to RANDOMIZED-PARTITON, the array is partitioned into two subarrays $A[\,p \ldots q-1\,]$ and $A[\,q+1 \ldots r]$, along with a ***pivot*** element $A[\,q\,]$.

- The elements of subarray $A[\,p \ldots q-1\,]$ are all $\leq A[\,q\,]$.
- The elements of subarray $A[\,q+1 \ldots r]$ are all $> A[\,q\,]$.
- The pivot elemen is kth element of subarray $A[\,p \ldots q]$, where $k = q - p + 1$.
- If the pivot element is the ith smallest element (i.e., $i = k$), return $A[\,q\,]$.
- Otherwise, recurse on the subarray containing the ith smallest element.
  - If $i < k$, this subarray is $A[\,p \ldots q-1\,]$, and we want the ith smallest element.
  - If $i > k$, this subarray is $A[\,q+1 \ldots r\,]$ and, since there are k elements in $A[\,p \ldots r]$ that precede $A[\,q+1 \ldots r\,]$, we want the $(i-k)$th smallest element of this subarray.

Analysis

Worst-case running time: $\Theta(n^2)$, because we could be extremely unlucky and always recurse on a subarray that is only 1 element smaller than the previous subarray.

Expected running time: RANDOMIZED-SELECT works well on average.

Analysis

The running time of RANDOMIZED-SELECT is a random variable $T(n) \Rightarrow$ to obtain $E[T(n)]$ as follows:

- For $k = 1, 2, \ldots, n$, define indicator random variable $X_k = I$ {subarray $A[p \ldots q]$ has exactly k elements}.

- Since Pr {subarray $A[p \ldots q]$ has exactly k elements} = $1/n$, Lemma 5.1 says that $E[X_k] = 1/n$.

Analysis

- Therefore, we have the recurrence

$$T(n) \le \sum_{k=1}^{n} X_k \cdot \left(T\left(\max(k-1, n-k)\right) + \mathrm{O}(n)\right)$$

$$= \sum_{k=1}^{n} X_k \cdot T\left(\max(k-1, n-k)\right) + \mathrm{O}(n).$$

- Taking expected values gives

$$\mathrm{E}\left[T(n)\right] \le \mathrm{E}\left[\sum_{k=1}^{n} X_k \cdot T\left(\max(k-1, n-k)\right) + O(n)\right]$$

$$= \sum_{k=1}^{n} \mathrm{E}\left[X_k \cdot T\left(\max(k-1, n-k)\right)\right] + O(n) \qquad \text{(linearity of expectation)}$$

$$= \sum_{k=1}^{n} \mathrm{E}\left[X_k\right] \cdot \mathrm{E}\left[T\left(\max(k-1, n-k)\right)\right] + O(n) \qquad \text{(equation (C.23))}$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot \mathrm{E}\left[T\left(\max(k-1, n-k)\right)\right] + O(n).$$

Analysis

- We rely on X_k and $T(\max(k-1, n-k))$ being independent random variables in order to apply equation (C.23).
- Looking at the expression $\max(k-1, n-k)$, we have
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation.
  - If n is odd, these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once.
- Either way, $\mathrm{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathrm{E}[T(k)] + O(n).$

Analysis

- Solve this recurrence by substitution:
  - Guess that $T(n) \le cn$ for some constant c that satisfies the initial conditions of the recurrence
  - Assume that $T(n) = O(1)$ for $n <$ some constant
  - Also pick a constant a such that the function described by the $O(n)$ term is bounded from above by an for all $n > 0$

Analysis

成功大學

$$\mathrm{E}\left[T(n)\right] \le \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} ck + an$$

$$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \right) + an$$

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{\left(\lceil n/2 \rceil - 1\right)\lceil n/2 \rceil}{2} \right) + an$$

$$\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{\left(n/2 - 2\right)\left(n/2 - 1\right)}{2} \right) + an$$

Analysis

$$
\begin{aligned}
&= \frac{2c}{n}\left(\frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an \\
&= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\le \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right).
\end{aligned}
$$

Analysis

- To complete this proof, we choose c such that

$$cn/4 - c/2 - an \geq 0$$

$$cn/4 - an \geq c/2$$

$$n(c/4 - a) \geq c/2$$

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

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

- Thus, as long as we assume that $T(n) = O(1)$ for $n < 2c/(c - 4a)$, we have $E[T(n)] = O(n)$.

Therefore, we can determine any order statistic in linear time on average.

Selection in worst-case linear time

SELECT works on an array of $n > 1$ elements:

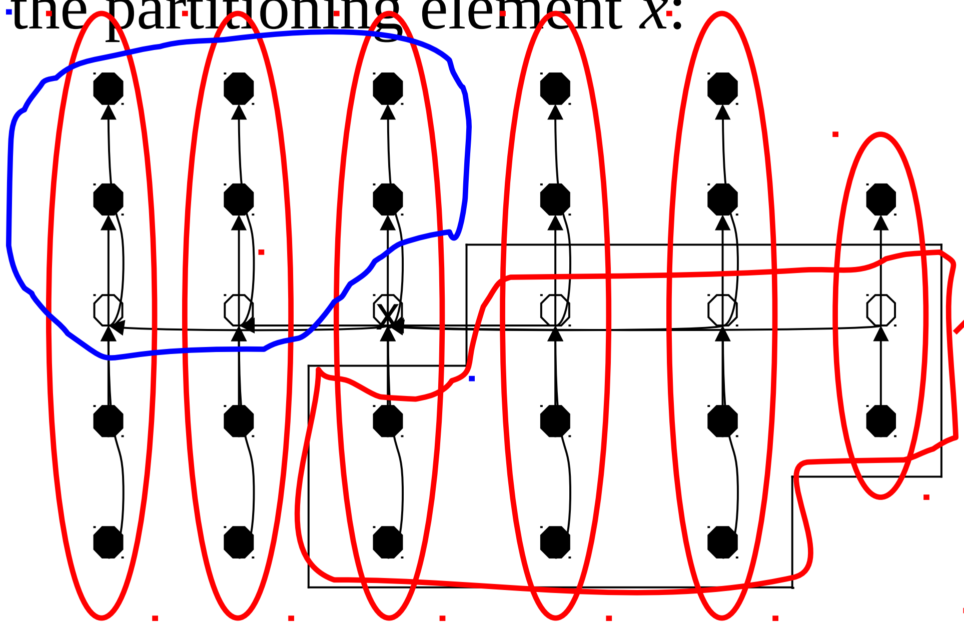
1. **Divide the *n* elements into groups of 5. Get $\lceil n / 5 \rceil$ groups: $\lfloor n / 5 \rfloor$ groups with exactly 5 elements and, if 5 does not divide *n*, one group with the remaining *n* mod 5 elements.**
2. **Find the median of each of the $\lceil n / 5 \rceil$ groups:**
   - ▷ **Run insertion sort on each group. Takes $O(1)$ time per group since each group has ≤ 5 elements.**
   - ▷ **Then just pick the median from each group, in $O(1)$ time.**
3. **Find the median *x* of the $\lceil n / 5 \rceil$ medians by a recursive call to SELECT. (If $\lceil n / 5 \rceil$ is even, then follow our convention and find the lower median.)**
4. **Using the modified version of PARTITION that takes the pivot element as input, partition the input array around *x*. Let *x* be the *k*th element of the array after partitioning, so that there are $k - 1$ elements on the low side of the partition and $n - k$ elements on the high side.**

Selection in worst-case linear time

5. Now there are three possibilities:
   - If $i = k$, just return x.
   - If $i < k$, return the ith smallest element on the low side of the partition by making a recursive call to SELECT.
   - If $i > k$, return the $(i - k)$th smallest element on the high side of the partition by making a recursive call to SELECT.

Analysis

Start by getting a lower bound on the number of elements that are greater than the partitioning element *x*:



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

[Each group is a column. Each white circle is the median of a group, as found in step 2. Arrows go from larger elements to smaller elements, based on what we know after step 4. Elements in the region on the lower right are known to be greater than x.]

Analysis

- At least half of the median found in step 2 are $\geq x$.
- Look at the groups containing these medians that are $\geq x$. All of them contribute 3 elements that are $> x$ (the median of the group and the 2 elements in the group greater than the group's median), except for 2 of the groups: the group containing x (which has only 2 elements $> x$) and the group with < 5 elements.
- Forget about these 2 groups. That leaves $\geq \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2$ groups with 3elements known to be $> x$.
- Thus, we know that at least

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

elements are $> x$.

Analysis

Symmetrically, the number of elements that are $< x$ is $\geq 3n / 10 - 6$.

Therefore, when we call SELECT recursively in step 5, it's on $\leq 7n / 10 + 6$ elements.

Develop a recurrence for the worst-case running time of SELECT:

- Steps 1, 2, and 4 each take $O(n)$ time:
  - Step 1: making groups of 5 elements takes $O(n)$ time.
  - Step 2: sorting $\lceil n / 5 \rceil$ groups in $O(1)$ time each.
  - Step 4: partitioning the n-element array around x takes $O(n)$ time.
- Step 3 takes time $T(\lceil n / 5 \rceil)$.
- Step 5 takes time $\leq T(7n / 10 + 6)$, assuming that $T(n)$ is monotonically increasing.

Analysis

- Assume that $T(n) = O(1)$ for small enough n. We'll use $n \le 140$ as "small enough."
- Thus, we get the recurrence

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

Solve this recurrence by substitution:

- ***Inductive hypothesis:*** $T(n) \le cn$ for some constant c and all $n > 0$.
- Assume that c is large enough that $T(n) \le cn$ for all $n \le 140$. So we are concerned only with the case $n \le 140$.
- Pick a constant a such that the function described by the $O(n)$ term in the recurrence is $\le an$ for all $n > 0$.

Analysis

- Substitute the inductive hypothesis in the right-hand side of the recurrence:

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

Analysis

- This last quantity is $\le cn$ if

$$
\begin{aligned}
-cn/10 + 7c + an &\le 0 \\
cn/10 - 7c &\ge an \\
cn - 70c &\ge 10an \\
c(n-70) &\ge 10an \\
c &\ge 10a(n/(n-70)).
\end{aligned}
$$

Analysis

- Because we assumed that $n \geq 140$, we have $n / (n - 70) \leq 2$.
- Thus, $20a \geq 10a (n / (n - 70))$, so choosing $c \geq 20a$ gives $c \geq 10a (n / (n - 70))$, which in turn gives us the condition we need to show that $T (n) \leq cn$.
- We conclude that $T (n) = O (n)$, so that SELECT runs in linear time in all cases.

Analysis

Notice that SELECT and RANDOMIZED-SELECT determine information about the relative order of elements only by comparing elements.

- Sorting requires $\Omega(n\lg n)$ time in the comparison model.
- Sorting algorithms that run in linear time need to make assumptions about their input.
- Linear-time *selection* algorithms do not require any assumptions about their input.
- Linear-time selection algorithms solve the selection problem without sorting and therefore are not subject to the $\Omega(n\lg n)$ lower bound.