Topic 5: Linear Time Selection

(CLRS 9)

CPS 230, Fall 2001

1 Selection

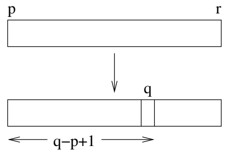
- If we could find element e such that $rank(e) = n/2$ (the *median*) in $O(n)$ time we could make quick-sort run in $\Theta(n \log n)$ time worst case.
  - We could just exchange e with first element in A in beginning of PARTITION and thus make sure that A is always partition in the middle.
- We will consider a more general problem than finding the ith element:
  - Selection problem

SELECT(i) is the ith element in the sorted order of elements

  - Note: We do not require that we sort to find SELECT(i)
  - Note: SELECT(1)=minimum, SELECT(n)=maximum, SELECT($\lceil n/2 \rceil$)=median.
- Special cases of SELECT(i):
  - Minimum or maximum can easily be found in $n-1$ comparisons
    - Scan through elements maintaining minimum/maximum.
  - Second largest/smallest element can be found in $(n-1)+(n-2) = 2n-3$ comparisons.
    - Find and remove minimum/maximum.
    - Find minimum/maximum.
  - Median:
    - Using the above idea repeatedly we can find the median in time $\sum_{i=1}^{n/2}(n-i) = n^2/2 - \sum_{i=1}^{n/2} i = n^2/2 - (n/2 \cdot (n/2+1))/2 = \Theta(n^2)$.
    - We can easily design $\Theta(n \log n)$ algorithm using sorting.
- Can we design $O(n)$ time algorithm for general i?
- If we could partition perfectly (which is what we are really trying to do) we could solve the problem
  - by partitioning and then recursively looking for the element in one of the partitions:

SELECT(A, p, r, i)

IF $p = r$ THEN RETURN $A[p]$
q=PARTITION(A, p, r)



$k = q - p + 1$
IF $i = k$ THEN RETURN $A[q]$
ELSE IF $i < k$ THEN
    RETURN SELECT$(A, p, q - 1, i)$
ELSE
    RETURN SELECT$(A, q + 1, r, i - k)$
FI

Select ith element using SELECT$(A, 1, n, i)$

– If the partition was perfect ($q = n/2$) we have

$$\begin{aligned} T(n) &= T(n/2) + n \\ &= n + n/2 + n/4 + n/8 + \cdots + 1 \\ &= \sum_{i=0}^{\log n} \frac{n}{2^i} \\ &= n \cdot \sum_{i=0}^{\log n} (\frac{1}{2})^i \\ &\leq n \cdot \sum_{i=0}^{\infty} (\frac{1}{2})^i \\ &= 2n. \end{aligned}$$

Note:

* The trick is that we only recurse on one side.
* In the worst case the algorithm runs in $T(n) = T(n-1) + n = \Theta(n^2)$ time.
* We could use randomization to get good partition on the average.
* Even if we just always partition such that a constant fraction ($\alpha < 1$) of the elements are eliminated we get running time $T(n) = T(\alpha n) + n = n \sum_{i=0}^{\log n} \alpha^i = \Theta(n)$.

- Rigorous average-case analysis: To simplify the analysis, we assume all inputs are distinct—in fact, we will assume that the array consists of the n distinct numbers 1 through n.
  – We assume that there are $n!$ distinct and *equally likely* input configurations.
  – q returned by PARTITION is the *rank* of $x = A[r]$ (i.e., the number of elements $\leq x$).
  – Probability that $rank(x) = q$ is $1/n$, for each $1 \leq q \leq n$.
  – Let $T_a(n)$ = average running time to find the ith element

- We can express the expected running time $T_a(n)$ as the weighted combination of n conditional expected running times, where each conditional expectation is conditioned on a different rank q for the partitioning element:

$$\begin{aligned}
T_a(n) &= \Theta(n) + \sum_{rank(x)=q} \mathrm{Prob}[rank(x)=q] \cdot (\text{expected running time given that } rank(x)=q) \\
&= \Theta(n) + \frac{1}{n} \cdot \sum_{rank(x)=q} (\text{expected running time given that } rank(x)=q) \\
&\le \Theta(n) + \frac{1}{n} \sum_{q=1}^{n} \max\{T_a(i-1), T_a(n-q)\} \qquad (1) \\
&= \Theta(n) + \frac{2}{n} \left(\sum_{i=\lceil n/2 \rceil}^{n-1} T_a(i) \right) + \frac{1}{n} T_a(\lfloor n/2 \rfloor)[n \text{ is odd}] \\
&\le \Theta(n) + \frac{2}{n} \sum_{i=\lfloor n/2 \rfloor}^{n-1} T_a(i)
\end{aligned}$$

- In the step leading to (1) we made the worst-case assumption that we're always unlucky in that the item we're selecting is in the larger of the two subarrays created by the partition. *This is an important point to make the analysis rigorous, because in the recursive calls, i may vary considerably.* The larger subarray clearly takes more time to process than the smaller subarray.
- Solution to

$$T_a(n) = n + \frac{2}{n} \sum_{i=\lfloor n/2 \rfloor}^{n} T_a(i) :$$

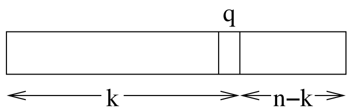
We prove $T_a(n) \le cn$. Induction step:

$$\begin{aligned}
T_a(n) &\le n + \frac{2}{n} \sum_{i=\lfloor n/2 \rfloor}^{n-1} ci \\
&= n + \frac{2c}{n} \left(\frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\le n + c(n-1) - \frac{c}{n} \left(\frac{n}{2} - 1 \right) \left(\frac{n}{2} - 2 \right) \\
&= n + c(n-1) - \frac{c}{n} \left(\frac{n^2}{4} - \frac{3n}{2} + 2 \right) \\
&= n + c \left(\frac{3}{4} n + \frac{1}{2} - \frac{2}{n} \right) \\
&\le \frac{3}{4} cn + \frac{c}{2} + n \\
&\le cn,
\end{aligned}$$

as long as $\frac{cn}{4} \ge \frac{c}{2} + n$, which we can make true for large enough n by choosing $c > 4$. We should also check the base cases to be sure that the hypothesis is true for the smaller values of n.

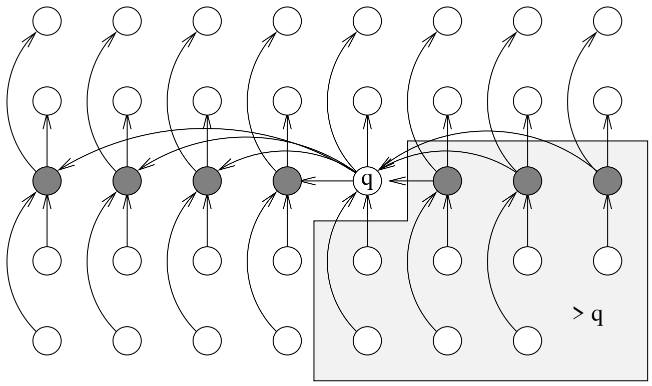
2 Efficient Selection in the Worst Case

- It turns out that we can modify the algorithm and get $T(n) = \Theta(n)$ in the worst case.
  - The idea is to find a split element q such that we always eliminate a fraction of the elements:

> SELECT(i)
> - Divide n elements into groups of 5.
> - Select median of each group ($\Longrightarrow \lceil \frac{n}{5} \rceil$ selected elements).
> - Use SELECT recursively to find median q of selected elements.
> - Partition *all* elements based on q.
>
> 
>
>
> - Use SELECT recursively to find ith element.
>   - If $i = k$, we're done.
>   - If $i < k$ then use SELECT(i) on $k - 1$ elements.
>   - If $i > k$ then use SELECT($i - k$) on $n - k$ elements.

  - If n' is the maximal number of elements we recurse on in the last step of the algorithm, the running time is given by $T(n) = \Theta(n) + T(\lceil \frac{n}{5} \rceil) + \Theta(n) + T(n')$

- Estimation of n':
  - Consider the following figure of the groups of 5 elements.
    - An arrow between element e_1 and e_2 indicates that $e_1 > e_2$.
    - The $\lceil \frac{n}{5} \rceil$ selected elements are drawn solid (q is median of these).
    - Elements that we *know for sure* are larger than or equal to q are indicated within the box.



  - The number of elements that we know for sure are larger than or equal to q is *at least*

$$\begin{aligned} 3(\#\text{ columns in shaded box}) - 1 - 2 &\geq 3\left(\frac{1}{2}\left\lceil \frac{n}{5} \right\rceil\right) - 3 \\ &\geq \frac{3n}{10} - 3. \end{aligned}$$

- - * This inequality is a little tighter than the one given in the book.
    * Justification: The number of columns in the box is at least $\frac{1}{2}\lceil\frac{n}{5}\rceil$. (The inequality is an equality when n is even.) Each column contributes three elements, except that the first contributes only two and the partially filled column (which by coincidence only happens to be the rightmost one in the picture) may contribute only one (e.g., if $n \bmod 5 = 1$).
  - Similarly the number of elements that we know for sure are less or equal to q is *at least* $\frac{3n}{10} - 3$.
    $\Longrightarrow$ We recurse on at most $n' = n - (\frac{3n}{10} - 3) = \frac{7}{10}n + 3$ elements
- So SELECTION(i) runs in time $T(n) = \Theta(n) + T(\lceil\frac{n}{5}\rceil) + T(\frac{7}{10}n + 3)$
- Solution to $T(n) = n + T(\lceil\frac{n}{5}\rceil) + T(\frac{7}{10}n + 3)$:
  - Guess $T(n) \leq cn$
  - Induction:

$$\begin{aligned}
T(n) &= n + T\left(\left\lceil\frac{n}{5}\right\rceil\right) + T\left(\frac{7}{10}n + 3\right) \\
&\leq n + c \cdot \left\lceil\frac{n}{5}\right\rceil + c\left(\frac{7}{10}n + 3\right) \\
&\leq n + c\frac{n}{5} + c + \frac{7}{10}cn + 3c \\
&= \frac{9}{10}cn + n + 4c \\
&\leq cn,
\end{aligned}$$

    if $4c + n \leq \frac{1}{10}cn$, which can be satisfied easily by choosing c large enough (at least 10) for sufficiently large n, and by checking by base conditions for the smaller values of n.
  - Note: It is important that we chose every 5th element, not all other choices will work (cf. homework).