Algorithms 07

CS201

Kaustuv Nag

Order Statistics, Medians, and Selection Problem

- The ith order statistic of a set of n elements is the ith smallest element.
  - The minimum of a set of elements is the first order statistic $(i = 1)$.
  - The maximum of a set of elements is the nth order statistic $(i = n)$.
- A median, informally, is the "halfway point" of the set.
  - When n is odd, the median is unique, occurring at $i = {}^{(n+1)}/_{2}$.
  - When n is even, there are two medians, occurring at $i = {}^{n}/_{2}$ and $i = {}^{n}/_{2} + 1$.

Selection Problem

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.

- We can solve the selection problem in $O(n \lg n)$ time, since we can sort the numbers using heapsort or merge sort and then simply index the ith element in the output array.
- Can we do better?

Finding Minimum and Maximum

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

- We can find the minimum with $(n-1)$ number of comparisons.
- We can find the maximum with $(n-1)$ number of comparisons.
- We can determine both the minimum and the maximum of n elements using $\theta(n)$ comparisons, which is asymptotically optimal: simply find the minimum and maximum independently, using $n-1$ comparisons for each, for a total of $2n-2$ comparisons.
  - Can we do better?
  - We can do it in using at most $3\lfloor n/2 \rfloor$ comparisons.

Simultaneously Finding Minimum and Maximum

- We do so by maintaining both the minimum and maximum elements seen thus far.
- We compare pairs of elements from the input first with each other, and then we compare the smaller with the current minimum and the larger to the current maximum, at a cost of 3 comparisons for every 2 elements.
- If n is odd, we set both the minimum and maximum to the value of the first element, and then we process the rest of the elements in pairs.
  - $\lfloor 3n/2 \rfloor$ comparisons.
- If n is even, we perform 1 comparison on the first 2 elements to determine the initial values of the minimum and maximum, and then process the rest of the elements in pairs as in the case for odd n.
  - $3(n-2)/2$ comparisons.

Selection in Expected Linear Time

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Selection in Expected Linear Time

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

Selection in Expected Linear Time: Analyzing the Algorithm

- Let the running time on an input array $A[p..r]$ of n elements be a random variable that we denote by $T(n)$, and we want to obtain an upper bound on $\mathbb{E}(T(n))$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- For each k, such that $1 \leq k \leq n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability $1/n$.
- For $k = 1, 2, \cdots, n$,we define indicator random variables X_k where

$$X_k = I\{\text{the subarray } A[p..q] \text{ has exactly } k \text{ elements}\}$$

- For a given call of RANDOMIZED-SELECT, the indicator random variable X_k has the value 1 for exactly one value of k, and it is 0 for all other k.
- Assuming that the elements are distinct, we have $\mathbb{E}[X_k] = 1/n$.

Selection in Expected Linear Time: Analyzing the Algorithm

- When $X_k = 1$, the two subarrays on which we might recurse have sizes $k-1$ and $n-k$. Hence, we have the recurrence

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

Selection in Expected Linear Time: Analyzing the Algorithm

- Taking expected values, we have

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

Selection in Expected Linear Time: Analyzing the Algorithm

- We have

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

- 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, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once.
- Thus, we have

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

Selection in Expected Linear Time: Analyzing the Algorithm

- We show that $\mathbb{E}[T(n)] = O(n)$ by substitution.
- We assume:
  - $\mathbb{E}[T(n)] \leq cn$ some constant c that satisfies the initial conditions of the recurrence.
  - $T(n) = O(1)$ for n less than some constant; we shall pick this constant later.
  - a is a constant such that the function described by the $O(n)$ term (which describes the non-recursive component of the running time of the algorithm) is bounded from above by an for all $n > 0$.
- Thus, we have

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

Selection in Expected Linear Time: Analyzing the Algorithm

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

Selection in Expected Linear Time: Analyzing the Algorithm

- We need to show that for sufficiently large n, this last expression is at most cn or, equivalently, that $cn/4 - c/2 - an \geq 0$.
- If we add $c/2$ to both sides and factor out n, we get $n(c/4 - a) \geq c/2$.
- 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

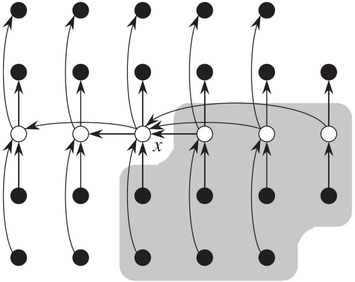
$$n \geq \frac{c/2}{c/4 - a} = \frac{2c}{c - 4a} .$$

- Thus, if we assume that $T(n) = O(1)$ for $n < 2c/(c - 4a)$, then $\mathbb{E}[T(n)] = O(n)$.

Selection in Worst-case Linear Time

- The following algorithm (we call it SELECT) determines the ith 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 ith smallest.)
  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 $\lceil n/5 \rceil$ groups by first insertion-sorting the elements of each group (of which there are at most 5) and then picking the median from the sorted list of group elements.
  3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians found in step 2. (If there are an even number of medians, then by our convention, x is the lower median.)
  4. Partition the input array around the median-of-medians x using the modified version of the partition method (used in quick sort). Let k be one more than the number of elements on the low side of the partition, so that x is the kth smallest element and there are $n - k$ elements on the high side of the partition.
  5. If $i = k$, then return x. Otherwise, use SELECT recursively to find the ith 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 Expected Linear Time: Analyzing the Algorithm



- The medians of the groups are whitened, and the median-of-medians x is labeled.
- Arrows go from larger elements to smaller, from which we can see that 3 out of every full group of 5 elements to the right of x are greater than x, and 3 out of every group of 5 elements to the left of x are less than x.
- The elements known to be greater than x appear on a shaded background.

Selection in Expected Linear Time: Analyzing the Algorithm

- The number of elements greater than x is at least

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

- Similarly, at least $3n/10 - 6$ elements are less than x.
- Thus, in the worst case, step 5 calls SELECT recursively on at most $7n/10 + 6$ elements.
- Thus, the runtime is

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

White Board

White Board