Australian National University
Research School of Computer Science

COMP3600/COMP6466 in 2016 – Tutorial Two

Solutions

Question 1. In the linear-time selection algorithm, the elements in the input sequence are divided into groups of size 5. Does the algorithm still run in linear time if the input elements are divided into groups of size 17 instead?

Yes, when the group size is 17, we can easily bound the sizes of R_1 and R_3, using similar discussions as we have done for group size of 5.

We assume that the ith smallest element that we are looking for is in R_3, we now show that the size of R_1 is bounded. Otherwise (i.e., the element is in R_1 or R_2), we can show that the size of R_3 is bounded as well.

$$|R_1| \geq 9(\frac{\lceil n/17 \rceil}{2} - 2) \geq \frac{9n}{34} - 18.$$

Thus, the size of R_3 is

$$|R_3| = n - |R_1| - |R_2| \leq n - |R_1| \leq \frac{25n}{34} + 18.$$

The time complexity is thus represented as the following recurrence.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 100 \\ T(\lceil n/17 \rceil) + T(25n/34 + 18) + O(n) & \text{if } n > 100 \end{cases}$$

We show that $T(n) = O(n)$. Since $25n/34$ is not always an integer, we have two choices. Either we rewrite the recurrence to use $T(\lceil 25n/34 \rceil + 18)$ instead of $T(25n/34+18)$ (we can do so since we assume that $T(n)$ is monotonically increasing), or we prove that there is some constant c such that $T(n) \leq cn$ for all rational numbers n (larger than some initial value n_0). (Note that an inductive proof for all n that are exact powers of 34 doesn't work here.) We prove the result for all positive rational numbers.

Choose c large enough that $T(n) \leq cn$ for $0 < n \leq 100$, and large enough that the same c works in the inductive step too. (This are the base cases.)

Also suppose the $O(n)$ term in the recurrence is bounded from above by an for $n > 100$.

Let n be any rational number greater than 100. We assume that for all n' such that $0 < n' < n$, we have $T(n') \leq cn'$. (This is the induction hypothesis.)

For n, we have

$$
\begin{aligned}
T(n) &\le T(\lceil n/17 \rceil) + T(25n/34 + 18) + an & (1)\\
&\le c\lceil n/17 \rceil + 25cn/34 + 18c + an & (2)\\
&\le cn/17 + c + 25cn/34 + 18c + an & (3)\\
&= 27cn/34 + 19c + an & (4)\\
&= cn + (-7cn/34 + 19c + an) & (5)\\
&\le cn & (6)
\end{aligned}
$$

as long as $-7cn/34 + 19c + an \le 0$. Since $n > 100$, this is satisfied for any $c \ge 63a$. So there is some large enough c that works in the base cases and in the inductive step too. Thus, we have $T(n) = O(n)$.

Question 2. Can you give an example that Quicksort takes $O(n^2)$ time? assuming that the first element in each sequence is used as the pivot element?

We assume that the n elements in a sequence is to be sorted in increasing order. If the input is a decreasing sequence, then the Quicksort takes $O(n^2)$ time if the first element in the sequence is chosen as the pivot element, as the use of the pivot element to partition the sequence with length $l > 1$ into two subsequences, one with length one; and another with length $(l-1)$, this takes $Ol)$ time. Thus, sorting all elements in the sequence takes $\sum_{l=2}^{n}(l-1) = O(n^2)$.

Question 3. To design of efficient algorithms for a problem $\mathcal{A}$, four people have the following conclusions:

- John showed that the problem takes $\Omega(n \log n)$ time.
- Amy devised an algorithm for the problem that takes $O(n^3)$ time.
- Lily proposed an algorithm for the problem that takes $o(n^2 \log^3 n)$ time.
- Peter proved that the problem takes $\omega(n^2)$ time.

Can you derive the tight bound of the problem? If not, what is the best possible range of its tight bound?

From the above arguments, we cannot derive the tight bound of the problem. But, following the conclusions of both John and Peter, a better lower bound of the problem is $\omega(n^2)$, while following the results by Amy and Lily, a better upper bound of the problem is $o(n^2 \log^3 n)$. The tight bound of the problem is between $\omega(n^2)$ and $o(n^2 \log^3 n)$.

Question 4.
Give an $O(n^2)$-time dynamic programming algorithm to find a longest decreasing subsequence of a sequence of n numbers.

Let $x_1, x_2, \ldots, x_n$ be the given sequence. The key property is: for any j, the longest decreasing sequence ending with x_j either consists only of x_j only or a longest decreasing sequence ending with some earlier x_i that is larger than x_j with $i < j$.

This gives a recurrence. Denote by $D(j)$ the length of the longest decreasing sequence ending with x_j, for $1 \leq j \leq n$. Then, for $1 \leq j \leq n$,

$$D(j) = \max_{i<j \& x_i > x_j} \{1, D(i) + 1\}.$$

Use the recurrence to compute $D(1), D(2), \ldots, D(n)$, in that order. Then the largest $D(j)$ is the answer to the problem. Clearly, the computation of $D(j)$ takes $O(j)$ time, in total, the algorithm takes $\sum_{j=1}^{n} O(j) = O(n^2)$ time.