Australian National University
Research School of Computer Science

COMP3600/COMP6466 in 2015 – Quiz Two

Due: 23:55pm Friday, August 14

Submit your work electronically through Wattle. The total mark of this quiz worths 10 points which is worth 3.5 of the final mark.

Question 1 (2 points).
(a) Assuming that there is a problem with problem size n, its lower bound on the running time is $\Omega(n^2 \log^{(6)} n)$, in the following definitions on optimal algorithms for the problem, which one is correct?

1. Its running time is the square of the problem size, e.g., $O(n^2)$
2. The lower bound of its running time is $\Omega(n^3 \log \log n)$
3. The upper bound of its running time is $O(n^{2.5} \log \log \log n)$
4. The difference between the lower bound and the upper bound on its running time is a constant factor, e.g., its lower bound is $\Omega(n^2 \log^{(6)} n)$, while its upper bound is $O(n^2 \log^{(6)} n)$ as well.

Answer: An algorithm for a problem is optimal if its worst running time (in the big-O sense) matches a lower bound of the problem, and the only difference between the upper bound and the lower bound of its running time is a constant coefficient. Thus, it is (4).

(b) In the following listed sorting algorithms, (1) which are optimal algorithms and which are not? (2) Express the running time of the optimal algorithm for sorting in the big-O notation, assuming that there are n elements to be sorted.

- bubble sort
- merge sort
- insertion sort
- quick sort
- shell sort

Answer: The optimal algorithm is the merge sort algorithm, the rest are not, as the running time of algorithm merge-sort is $\Theta(n \log n)$ if there are n elements to be sorted.

Question 2 (2 points).
(a) Given a problem $\mathcal{A}$ with problem size n, student Bob shows its lower bound is $n \log n$, while student Alice proves that its lower bound is $\sqrt{n} \log^3 n$. Whose solution is better in terms of the quality of the solutions? Justify your answer.

Answer: Bob's solution is better than Alice's one in terms of the solution quality, as the estimated accuracy of Bob's solution is more accurate (to the exact solution – the tight bound) than that of Alice's solution.

(b) Given a problem $\mathcal{A}$ with problem size n, student X devised an algorithm for it with time complexity $O(n^{3/2} \log n)$, student Y proposed an algorithm for the problem with time complexity $O(n^2 \log \log n)$, and student Z devised an algorithm for it with time complexity $O(n^{1.2} \log \log n)$, assuming a lower bound of problem $\mathcal{A}$ is $\Omega(n \log n)$. Among these three algorithms, which one is the best? Order the three algorithms in terms of their running times in increasing order? Justify your answer.

Answer: The solution provided by student Z is the best one as the upper bound given by Z is the nearest one to a lower bound of the problem $\Omega(n \log n)$ among the three solutions. The other two do not.

Following the time complexity analysis of the three algorithms, the running times of these three algorithms in increasing order are : Z's algorithm, X's algorithm, and Y's algorithm, as the estimate (over estimation to the exact running time) of Z's solution is more accurate (to the tight bound) than those of X and Y.

Question 3 (3 points).
In the linear-time selection algorithm, the n elements in the input sequence are divided into groups of size 5 except the last group. Does the algorithm still run in time $O(n)$ if the input elements are divided into groups of size 71 instead? Formulate the time complexity of the algorithm as a recurrence and solve the recurrence.

Answer: 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 lower bounded. Otherwise (i.e., the i-th smallest element is in R_1 or R_2), we can show that the size of R_3 is upper bounded as well.

$$|R_1| \geq \frac{71+1}{2}(\frac{\lceil n/71 \rceil}{2} - 2) \geq 36(\frac{n}{2 \times 71} - 2) = \frac{18n}{71} - 72.$$

Thus, the size of R_3 is

$$|R_3| = n - |R_1| - |R_2| \leq n - |R_1| \leq n - (\frac{18n}{71} - 72) = \frac{53n}{71} + 72.$$

The time complexity is thus represented as the following recurrence.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 71 \\ T(\lceil n/71 \rceil) + T(\frac{53n}{71} + 72) + O(n) & \text{if } n > 71 \end{cases}$$

We can prove by the method of substitution that for any rational number $n > 0$, $T(n) \leq bn$ for some constant $b > 0$. Choose b large enough that $T(n) \leq bn$ for all $n \leq 71$.

Also suppose the $O(n)$ term in the recurrence is bounded by an for $n > n_0$.

For $n > n_0$, the recurrence says

$$\begin{aligned} T(n) &\leq T(\lceil n/71 \rceil) + T(\frac{53n}{71} + 72) + an && (1) \\ &\leq b\lceil n/71 \rceil + (b\frac{53n}{71} + 72b) + an && (2) \\ &\leq (bn/71 + b) + \frac{53bn}{71} + 72b + an && (3) \\ &= \frac{54bn}{71} + 73b + an && (4) \\ &= (bn - \frac{17bn}{71}) + 73b + an && (5) \\ &= bn + (-\frac{17bn}{71} + 73b + an) && (6) \\ & && (7) \end{aligned}$$

The expression $-\frac{17bn}{71} + 73b + an$ is non-positive if $n \geq n_0$ and $b \geq \frac{71an}{73 \times 71 + 17n} = \frac{71a}{73+17}$ when $n_0 = 71$. Therefore, for some b, $T(n) \leq bn$ always.

Question 4 (3 points).
A subsequence is *palindromic* if it is the same whether read from left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is not palindromic).

Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence, and analyze the running time of the proposed algorithm.

Answer: Let $X = \langle x_1, x_2, \ldots, x_n \rangle$ be the given sequence. A subsequence is palindromic if it is the same whether read from left to right or from right to left.

Observe that a sequence Y is palindrome if we form a sequence Y' by reversing the sequence Y then $Y = Y'$. Following this observation we formulate an optimal substructure for the problem. If we examine the subsequence $X_{i..j} = x_i, x_{i+1}, \ldots, x_j$ of X, then we can find a palindrome of length at least 2 when $x_i = x_j$. If $x_i \neq x_j$, then we seek the maximum length palindrome in subsequences $X_{i+1..j}$ and $X_{i..j-1}$. Also every x_i is a palindrome in itself of length 1. Therefore, the base case is given by $x_{i..i} = x_i$. Let us define the maximum length palindrome for the subsequence $X_{i..j}$ as $L(i, j)$, then

$$L(i,j) = \begin{cases} 1 & i = j \\ 0 & i > j \\ L(i{+}1, j{-}1) + 2 & \text{if } x_i = x_j \\ \max\{L(i{+}1, j), L(i, j-1)\} & \text{otherwise} \end{cases}$$

Based on the above recursive definition, we have the following algorithm.

Procedure `Palindromic_subsequence`$(x[1, ..., n])$

Input: Sequence $x[1, .., n]$
Output: Length of the longest palindromic subsequence
for $i \leftarrow 1$ **to** n **do**
$L[i, i] \leftarrow 1$;
endfor;
for $i \leftarrow 1$ **to** n **do**
for $l \leftarrow 1$ **to** $n - i$ **do**
$j \leftarrow l + i$;
$L[l, j] \leftarrow Cost(L, x, l, j)$;
$L[j, l] \leftarrow Cost(L, x, j, l)$;
endfor;
endfor;
return $L[1, n]$;

The procedure `Cost` is an implementation of recursive formulation shown above.

Procedure `Cost`(L, x, i, j)

Input: Array L from procedure palindromic-subsequence; Sequence $x[1, ..., n]$, i and j are indices
Output: cost of $L[i, j]$ and the index array s;
if $i = j$ **then**
$s[i, i] \leftarrow i$;
return $L[i, j]$;
else
if $x_i = x_j$ **then**

```
            if      i + 1 < j − 1 then
                    return L[i + 1, j − 1] + 2
            else
                    return 2
            endif
        else
            return max(L[i + 1, j], L[i, j − 1]);
        endif
endif
```

Time complexity analysis: First **for** loop takes $O(n)$ time while the second **for** loop takes $O(n-i)$ too. Thus, the total running time of the proposed algorithm is $O(n^2)$.