Homework 5

Problem 1. Prove the following corollary of the time hierarchy theorem: for all constants $c_1 > c_0 \geq 1$, $\mathrm{TIME}(n^{c_0}) \subsetneq \mathrm{TIME}(n^{c_1})$.

Solution. First of all, we have the following lemma:

> **Lemma.** For all constants $c_1 > c_0 \geq 1$, $n^{c_0} = o(n^{c_1}/\log n)$.
>
> **Proof.** Since $c_1 > c_0 \geq 1$, we have
>
> $$\lim_{n\to\infty} \frac{n^{c_0}}{n^{c_1}/\log n} = \lim_{n\to\infty} \frac{\log n}{n^{c_1-c_0}} = 0.$$
>
> Therefore, $n^{c_0} = o(n^{c_1}/\log n)$.

From the hierarchy theorem, we have $\mathrm{TIME}(o(n^{c_1}/\log(n^{c_1}))) \subsetneq \mathrm{TIME}(n^{c_1})$. That is to say, $\mathrm{TIME}(o(n^{c_1}/\log n)) \subsetneq \mathrm{TIME}(n^{c_1})$. $n^{c_0} = o(n^{c_1}/\log n)$ from the lemma above, therefore, we have $\mathrm{TIME}(n^{c_0}) \subsetneq \mathrm{TIME}(n^{c_1})$.

Problem 2.

(a) Show that P is closed under union, concatenation, and complement. That is, $A \cup B, A \circ B, A^{\mathrm{c}} \in \mathrm{P}$ if $A, B \in \mathrm{P}$. Note that the concatenation $A \circ B$ of two languages A and B is defined as

$$A \circ B = \{xy \mid x \in A, y \in B\}.$$

(b) Show that P is closed under the star operation. That is, $A^* \in \mathrm{P}$ if $A \in \mathrm{P}$ where $A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, x_j \in A \text{ for } j = 1, 2, \ldots, k\}$. (Hint: You may need to use dynamic programming to maintain a table whose i, j-th entry indicates whether $x_i \cdots x_j \in A^*$)

Solution. (a) The proof is as follows:

1. $A \cup B$ is in P, because we can decide whether $x \in A \cup B$ by first checking whether $x \in A$ and then checking whether $x \in B$. If either of them is true, then $x \in A \cup B$. Since we can solve both A and B in polynomial time using deterministic Turing machines, so first solving A then B consequently will also be in polynomial time. Therefore, $A \cup B \in \mathrm{P}$.

2. $A \circ B$ is in P, because we can decide whether $x \in A \circ B$ by checking whether $x = yz$ where $y \in A$ and $z \in B$. We can iterate i from 0 to $|x|$ and check whether $x[0 \dots i-1] \in A$ and $x[i \dots |x|-1] \in B$. Since we can solve both A and B in polynomial time using deterministic Turing machines, $A \circ B \in \mathrm{P}$.

3. A^{c} is in P, because we can decide whether $x \in A^{\mathrm{c}}$ by by negating the result of $x \in A$. Since we can solve A in polynomial time using deterministic Turing machines, so solving A^{c} will also be in polynomial time. Therefore, $A^{\mathrm{c}} \in \mathrm{P}$.

(b) We can use dynamic programming to maintain a table whose i, j-th entry indicates whether $x_i \cdots x_j \in A^*$. The algorithm is as follows:

1. Initialize the table T with $T_{i,i} = \begin{cases} \text{true}, & \text{if } x_i \in A, \\ \text{false}, & \text{if } x_i \notin A \end{cases}$ for all i.

2. Iterate l from 1 to $|x|$.

3. Iterate i from 1 to $|x| - l$.

4. Set $j = i + l$.

5. Iterate k from i to $j - 1$.

6. Set $T_{i,j} = T_{i,j} \vee (T_{i,k} \wedge T_{k+1,j})$.

7. Check if $T_{1,|x|}$. If it is true, then $x \in A^*$; otherwise, $x \notin A^*$.

The correctness of the algorithm is obvious, since $x \in A^*$ ensures a possible partition of x into $x_1 \cdots x_k$ where $x_i \in A$ for all i, and the algorithm will find such a partition; and the algorithm will only find such x because of the way the table is updated, which ensures that $T_{i,j}$ is true if and only if it can be partitioned into smaller segments $x_i \cdots x_j \in A^*$. Beside the initialization, the time complexity of the algorithm is $O(n^3)$, where n is the length of x. And the initialization can be done in polynomial time because A is in P. Therefore, $A^* \in \mathrm{P}$.

Problem 3. Karatsuba algorithm is an efficient algorithm for multiplying two natural numbers of $n = 2^k$ bits, outperforming the straightforward $O(n^2)$ primary-school

method. The key idea is as follows. First, we write the numbers as $a2^\ell + b$ and $c2^\ell + d$ where $\ell = n/2$ and $a, b, c, d \in \{0, 1, \ldots, 2^\ell - 1\}$. So the product is

$$ac2^{2\ell} + (ad + bc)2^\ell + bd,$$

and this reduces the computation of the product to four multiplications (ac, ad, bc, bd) of shorter numbers. Second, Karatsuba's key idea is that three multiplications suffice for the computation as one can first compute ac, bd. Then the coefficient in front of 2^ℓ can be computed by one extra multiplication as $ad + bc = (a + b)(c + d) - ac - bd$. Show that Karatsuba's algorithm has time complexity $O(n^{\log_2 3}) \approx O(n^{1.585})$.

Solution. Let $T(n)$ be the time complexity of Karatsuba's algorithm for multiplying two n-bit numbers. The problem is divided into three subproblems, which involve multiplying three pairs of numbers of the size $\lceil n/2 \rceil$. In addition, the time complexity of the 'conquer' part is linear to n, since the additions, subtractions, and digit shifts can be done in linear time. Then we have the following recurrence relation:

$$T(n) = 3T(\lceil n/2 \rceil) + O(n).$$

From the master theorem, we have $a = 3, b = 2, f(n) = O(n)$, and $n^{\log_b a} = n^{\log_2 3} \approx n^{1.585}$. Since $f(n) = O(n) = O(n^{\log_2 3 - \epsilon})$ for some $\epsilon > 0$, we have $T(n) = O(n^{\log_2 3}) \approx O(n^{1.585})$.