# Solutions:

1. The recursion is $M(n) = 7M(n/2) + f(n)$ where $f(n) = \Theta(n^2)$. We compare $f(n)$ with $n^{\log_2 7}$. Since $\log_2 7 \approx 2.80735$, we have $f(n) = O(n^{\log_2 7 - 0.007})$ and Case 1 of the Master Theorem applies. Therefore, $M(n) = \Theta(2^{\log_2 7}) \approx \Theta(n^{2.80735}$, which is better than $\Theta(n^3)$ of the naïve algorithm.

2. $T(n) = T(\lfloor \frac{n}{2} \rfloor)+$constans. The first addend is because we cut the array in at least halves. The second addend is the cost of cutting, which is independent of the size of the array.

3. We apply the Master Theorem.

   (a) Since $\log_3 3 = 1$, we compare $n^1$ with $n^{\frac{1}{2}}$. Since $f(n) \in O(n^{1-\frac{1}{4}})$, we apply Case 1 of the Master Theorem to get $T(n) = \Theta(n)$.

   (b) Since $\log_{10} 2 \approx 0.30103$, we compare $n^{\log_{10} 2}$ with $\sqrt{n}$. Since

   $$\sqrt{n} = n^{0.5} \in \Omega(n^{\log_{10} 2 + 0.1}),$$

   and since

   $$2f(n/10) = 2\sqrt{\frac{n}{10}} = \frac{2}{\sqrt{10}}\sqrt{n} = \frac{2}{\sqrt{10}}f(n)$$

   shows that the regularity condition is fulfilled, we are in Case 3 of the Master Theorem and obtain

   $$T(n) \in \Theta(\sqrt{n}).$$

   (c) Since $\log_3 3 = 1$, we compare $n$ with $n/3$. Obviously, $n \in \Theta(n/3)$, so we are in Case 2 and $T(n) = \Theta(n \log n)$.

   (d) Since $\log_4 2 = \frac{1}{2}$, we compare $n^{1/2}$ with $\sqrt{n} \log n$. Because

   $$\frac{\sqrt{n} \log n}{\sqrt{n}} = \log n \longrightarrow_{n \to \infty} \infty$$

we have $\sqrt{n}\log n \in \Omega(n^{1/2})$, but we need more for Case 3, namely that $\sqrt{n}\log n \in \Omega(n^{1/2+\epsilon})$. However,

$$\frac{\sqrt{n}\log n}{n^{1/2+\epsilon}} = \frac{\log n}{n^{\epsilon}} \longrightarrow_{n\to\infty} 0,$$

so this is not the case: $\sqrt{n}\log n \notin \Omega(n^{1/2+\epsilon})$. We are therefore in the boundary between Case 2 and Case 3 and cannot apply the Master Theorem.

4. Sorting with merge-sort has worst case runtime of $\Theta(n \log n)$. Listing the $i$ largest numbers takes $i$ steps and is obviously $O(n)$. Therefore, the worst-case run time is $\Theta(n \log n)$. The selection algorithm takes worst time $\Theta(n)$ and the partition takes $n-1$ comparisons and hence also has worst time $\Theta(n)$. Therefore, the worst time is now $\Theta(n)$. Notice that we do **not** order the $i$ largest elements.

5. This algorithm is not correct. The second-largest element could be the looser in a match-up against the largest element and be therefore not in the list. However, the recursion would be $T(n) = T(\lceil \frac{n}{2} \rceil)+$constans, which would give us logarithmic time.

6. We use the limit of quotients.

   (a) $\dfrac{n(\log n)^2}{n \log n} = \log n \longrightarrow_{n\to\infty} \infty$ implies $n \log n \in o(n(\log n)^2)$.

   (b) We look at the logarithms of the functions. $\log(n^n) = n \log(n)$ and $\log(2^n) = \log 2 \cdot n$. Therefore,

   $$\lim_{n\to\infty} \log(\frac{2^n}{n^n}) = \lim_{n\to\infty} \Big(\log(2^n) - \log(n^n)\Big) = \lim_{n\to\infty} n(2 - \log(n)) = -\infty,$$

   which implies $\lim\limits_{n\to\infty} \dfrac{2^n}{n^n} = 0$, which in turn implies $n^n \in \Omega(2^n)$.

   (c) Because $\log_2(2^{(2^n)}) = 2^n$ and $\log_2(2^2)^n = n \log_2 2^2 = 2n$,

   $$\log_2(\frac{(2^2)^n}{2^{2^n}}) = \log_2((2^2)^n) - \log_2(2^{2^n}) = 2n - 2^n \longrightarrow_{n\to\infty} -\infty,$$

   implies that $\lim\limits_{n\to\infty} \dfrac{(2^2)^n}{2^{2^n}} = 0$ and therefore $(2^2)^n \in o(2^{2^n})$.

7. We concatenate the original string:

   (a) $00111000 = 0.01.1.0.0.0 \in 0^*(01)^*(0+1)^*$

   (b) $00111000 = 0.0.1.1.1.0.0.0 \in 0^*1^*0^*$

   (c) The regular expression only contains strings that are concatenations of two letter combinations. Since $00111000 = 00.11.10.00$ is the only decomposition possible, we see that it cannot fit the regular expression.

8. We list the set of states that can be obtained using 0- and 1-transitions. Since there is an epsilon-transition out of the start state, the beginning state set is {A, B}.

| State | 0 | 1 |
|---|---|---|
| {A, B} | {A,B} | {A, B, C} |
| {A, B, C} | {A, B, C} | {A, B, C} |

This gives us