Advanced Algorithms
Freie Universitat Berlin
Winter 2023-24
László Kozma & Michaela Krüger

Exercise Sheet 1

Jatin Kansal, Sandip Sah

This homework answers the problem set sequentially.

Exercise 1

Solution (a)

A general polynomial of degree k can be written in the form of a sum. So, we will use $f(n) = \sum_{i=0}^{k} a_i n^i$ everywhere for this problem (a_i are constants).

1. For the relation $f(n) \in \theta(n^k)$ to hold, it is sufficient to show that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ holds, where c is some constant. Here $g(n) = bn^k$ is a function representing the family of functions in $\theta(n^k)$. We can calculate this ratio:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sum_{i=0}^{k} a_i n^i}{bn^k} = \lim_{n\to\infty} \frac{n^k \sum_{i=0}^{k} a_i n^{k-i}}{bn^k} = c$$

   Since the extended relation holds, this implies $f(n) \in \theta(n^k)$

2. For the relation $f(n) \in o(n^l); l > k$ to hold, it is sufficient to show that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ holds. Here $g(n) = bn^l$ is a function representing the family of functions in $o(n^l)$. We can calculate this ratio:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sum_{i=0}^{k} a_i n^i}{bn^l} = \lim_{n\to\infty} \frac{\sum_{i=0}^{k} a_i n^{k-i}}{bn^{l-k}} = 0$$

   Since the extended relation holds, this implies $f(n) \in o(n^l)$

3. For the relation $f(n) \in \omega(n^l); l < k$ to hold, it is sufficient to show that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ holds. Here $g(n) = bn^l$ is a function representing the family of functions in $\omega(n^l)$. We can calculate this ratio:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sum_{i=0}^{k} a_i n^i}{bn^l} = \lim_{n\to\infty} b^{-1} n^{k-l} \sum_{i=0}^{k} a_i n^{k-i} = \infty$$

   Since the extended relation holds, this implies $f(n) \in \omega(n^l)$

Solution (b)

Consider the following piece-wise function

$$f(n) = \begin{cases} n^5 & \textbf{if n is odd} \\ n^3 & \textbf{if n is even} \end{cases}$$

Also consider $o(n^4)$. In this case $f(n) \notin o(n^4)$ as for odd values of n when n is sufficiently large, $f(n) > cn^4$ irrespective of the constant chosen. But, at the same time, $f(n) \notin \Omega(n^4)$ as for even values of n when n is sufficiently large, $f(n) < cn^4$ irrespective of the constant chosen. From this example, it is clear that $f(n) \notin o(g)$ does not imply $f(n) \in \Omega(g)$. The simple reason for this is that numbers cannot be discontinuous but functions can be.

Solution (c)

Function	Asymptotic function	Explanation
$n^{(1-\epsilon)}$	$O(n^{(1-\epsilon)})$	Since ϵ cannot be smaller than 0 or larger than 1, $n^{(1-\epsilon)}$ is a decaying function as the power is less than 1.
$3^{\sqrt{\log n}}$	$O(n^c); c < 1$	$3^{\sqrt{\log n}} = \frac{3^{\log n}}{3^{\sqrt{\log n}}} = \frac{n^k}{3^{\sqrt{\log n}}} < O(n^c); k < 1$ The above expression implies the function $3^{\sqrt{\log n}}$ is a decay function but the power is hard to determine.
$\frac{n}{\log(\log n)}$	$\theta(\frac{n}{\log(\log n)})$ or $O(n^c); c < 1$	We can't simplify this any further but given that for sufficiently large n, the denominator is greater than 1, this function is smaller than O(n) functions.
ϵn	$\theta(n)$	Self evident, ignore constant
$n^{\frac{3}{2}} \log n$	$\theta(n^{\frac{3}{2}} \log n)$	This function definitely grows faster than $O(n)$ as there is another multiplication factor alongside it. But, comparing this to $n^{8/5}$, we get that the growth rate of $\log(n)$ must equal growth rate of $n^{1/10}$ for those two functions to have the same asymptotic. But $\log(n)$ grows lower than $n^{1/10}$. That's why we placed $n^{\frac{3}{2}} \log n$ before $n^{\frac{8}{5}}$.
$n^{\frac{8}{5}}$	$\theta(n^{\frac{8}{5}})$	Same reasoning as above, but additionally, $n^{8/5}$ is clearly below n^2.
n^2	$\theta(n^2)$	The power of n stays constant whereas in the later functions, the power keeps increasing, no matter how slowly. For a large enough n, they will overtake n^2.
$n^{\log(\log n)}$, $(\log n)^{\log n}$	$\theta(n^{\log(\log n)})$	These two functions are mathematically equivalent that's why we put them together. Additionally, $\log(\log(n))$ will always be smaller than $0.5 \log(n)$ for sufficiently large n. Thus, the functions in this row will have a lower asymptotic bound than $n^{0.5 \log(n)}$
$n^{\frac{1}{2} \log n}$	$\theta(n^{\frac{1}{2} \log n})$	Even though both $n^{\frac{1}{2} \log n}$ and $n^{\sqrt{n}}$ are exponential functions, the power $\frac{1}{2} \log n$ will always grow slower than $\sqrt{n}$. As n grows sufficiently large, the $n^{\sqrt{n}}$ will take over.
$n^{\sqrt{n}}$	$\theta(n^{\sqrt{n}})$	Same reason as above. As n grows sufficiently large, $(1+\epsilon)^n$ will take over irrespective of the fact that the base of $n^{\sqrt{n}}$ also grows in value.
$(1+\epsilon)^n$	$\theta((1+\epsilon)^n)$	See reasons above.

Exercise 2

Probability of success on each experiment $= \frac{1}{\ln n}$
Probability of failure on each experiment $= 1 - \frac{1}{\ln n}$

Let's imagine doing two successive independent experiments.
Total probability of success $= \frac{1}{lnn} + \left(1 - \frac{1}{lnn}\right)\frac{1}{\ln n}$
Total probability of failure $= \left(1 - \frac{1}{\ln n}\right)^2$
This means, we could also write the total probability of success as $1 - \left(1 - \frac{1}{\ln n}\right)^2$
Similarly, for k successive independent experiments, our total probability of success can be written as $1 - \left(1 - \frac{1}{\ln n}\right)^k$
We want this probability to be at least $1 - \frac{1}{n^2}$, i.e., we want to find the value of k that satisfies the following inequality:

$$1 - \frac{1}{n^2} \leq 1 - \left(1 - \frac{1}{\ln n}\right)^k$$

So, lets solve this inequality. First, let's remove the constants and flip the signs:

$$\left(1 - \frac{1}{\ln n}\right)^k \leq \frac{1}{n^2}$$

Next, we use the binomial expansion of the left hand side to write the inequality $1 - \frac{k}{\ln n} \leq \left(1 - \frac{1}{\ln n}\right)^k$. We can only write this because $\frac{1}{lnn} \leq 1$. We use this inequality in the above inequality.

$$1 - \frac{k}{\ln n} \leq \frac{1}{n^2}$$

We can further simplify this to write:

$$k \geq \ln n\left(1 - \frac{1}{n^2}\right)$$

This is the best we could do to simplify the required value of k.

Exercise 3

(a)

Consider the following algorithm:

$i \leftarrow 2$
$A_1 \leftarrow A$
while $i \leq \lceil \frac{N}{2} \rceil$ **do**:
  $A_i \leftarrow A_{i/2}^2$
  $i \leftarrow 2i$

end while
$A^N \leftarrow \Pi_i A_i^{x_{\log(i)}}$

Here, the $x_{\log(i)}$ is either 1 or 0 and denotes the $\log(i)$th bit in the binary representation of N. What we are doing here constructing a binary basis using powers of A. We only need $\log(N)$ terms in this basis to then combine them to form A^N in a way that no term is needed more than once. For such an algorithm, it only requires $\log(N) + 2$ elementary steps because we can access the binary representation of N in our model without any additional computation. Hence, we can compute A^N in $O(\log(N))$ time.

(c)

We assume the result of part (b) that we can compute the binomial factor $\binom{N}{K}$ in $O(\log N)$ time. Assume the function to do this process be called ComputeBinom(N, K). Then, consider the following algorithm:

function ComputeFactorial(N)
 $N_{1/2} \leftarrow \frac{N}{2}$
 $N_K \leftarrow ComputeBinom(N, N_{1/2})$
 $N_{1/2,fac} \leftarrow ComputeFactorial(N_{1/2})$
 $N_{fac} \leftarrow N_K \times N_{1/2,fac}^2$
 return N_{fac}
end function

The recurrence function for the above algorithm can be written as:

$$T(N) = 2T(N/2) + O(\log N)$$

To solve this recurrence, let's think of constructing a binary tree. At each level of the tree, we incur the cost $O(\log N)$ (in reality this is a very loose bound, but sufficient for us). The tree will have $\log N$ levels because we split the problem in half at each level. So, the total run time complexity is $\log N \times O(\log N) = O(\log^2 N)$

(d)

If N is a prime number, then the only factors it has are N and 1. This means that none of the numbers from 2 to N-1 can divide N. Conversely, if we multiply all the numbers from 2 to N-1, then N cannot divide the resulting number. We just showed in part (c) that we can compute $N!$ for any given N in $O(\log^2 N)$ time. So, we can also compute $(N-1)!$ in $O(\log^2 N)$ time. Lastly, we divide $(N-1)!$ by N. If we get a whole number, then we conclude N is not prime but otherwise it is prime. And since division is an elementary process, our total running time is still denoted by $O(\log^2 N)$.

(e)

For any number K, if N divides $K!$, then all the prime factors of N can be found in $K!$ but if doesn't divide $K!$, then not all factors can be found in $K!$. So, consider the following algorithm:

function ComputeFactors(N)
 $i \leftarrow \frac{N}{2}$
 while 1 **do**

```
        i_fac ← ComputeFactorial(i)
        if N | i_fac then
            i ← i/2
        else
            div ← ComputeFactorial(2i)/i_fac
            factor ← GCD(div, N)
            break
        end if
    end while
    return factor
end function
```

For this algorithm, each loop requires $O(\log^2 N)$ running time in the worst case scenario. Moreover, the algorithm would only loop through at most $\log N$ times in the worst case scenario. Hence, the total runtime for such an algorithm is $\log N \times O(\log^2 N) = O(\log^3 N)$.