Exercises week 1
Answers

Complexity 2011-2012

A. Silva, H. Barendregt, B. Westerbaan & B. Westerbaan

Exercise 1. (*) Find f, g such that $f \not\leq g$ and $g \not\leq f$.

Answer Define

$$f(n) = \begin{cases} 1 & n \text{ is even} \\ 0 & n \text{ is odd} \end{cases} \quad \text{and} \quad g(n) = \begin{cases} 0 & n \text{ is even} \\ 1 & n \text{ is odd.} \end{cases}$$

Recall that $f \leq g$ by definition if

$$\exists_{c \in \mathbb{R}_{>0}} \exists_{N \in \mathbb{N}} \forall_{n \geq N} 0 \leq f(n) \leq cg(n).$$

Thus $f \not\leq g$ if

$$\forall_{c \in \mathbb{R}_{>0}} \forall_{N \in \mathbb{N}} \exists_{n \geq N} [f(n) < 0 \text{ or } cg(n) < f(n)].$$

But for any $c \in \mathbb{R}_{>0}$ and $N \in \mathbb{N}$ we have

$$cg(2N) = 0 < 1 = f(2N).$$

Thus $f \not\leq g$. Conversely,

$$cf(2N+1) = 0 < 1 = g(2N+1).$$

Thus also $g \not\leq f$.

Exercise 2. Show that

(i) $2^{n+1} \in O(2^n)$.

(ii) $2^{2n} \notin O(2^n)$.

(iii) $n^{10} \in O(2^n)$. (**Note:** this is also a harder exercise, we shall discuss it in class.)

Answers

(i) We need to show

$$\exists_{c \in \mathbb{R}_{>0}} \exists_{N \in \mathbb{N}} \forall_{n \geq N} 0 \leq 2^{n+1} \leq c2^n.$$

Note that $0 \leq 2^{n+1} = 2 \cdot 2^n$. Thus $c = 2$ and $N = 0$ will do.

(ii) We need to show
$$\forall_{c\in\mathbb{R}_{>0}}\forall_{N\in\mathbb{N}}\exists_{n\geq N}[2^{2n} < 0 \text{ or } c2^n < 2^{2n}].$$
Thus given c and N, its sufficient to find a $n > N$ such that $c < 2^n$. But such an n always exists! For instance, you can use: $n = \max\{\log_2 c + 1, N\}$.

(iii) By trying out, we find that $n^{10} \leq 2^n$ for $n = 100$ By induction, we will prove that this also holds for all $n \geq 100$.

Thus, suppose (induction hypothesis, IH) that $n^{10} \leq 2^n$ and $n \geq 100$. We want to prove $(n+1)^{10} \leq 2^{n+1}$. From IH we deduce: $2n^{10} \leq 2\cdot 2^n = 2^{n+1}$. If n is large enough such that $n+1 \leq \sqrt[10]{2}n$, then we are OK, because then: $(n+1)^{10} \leq (\sqrt[10]{2}n)^{10} = (\sqrt[10]{2})^{10}n^{10} = 2n^{10} \leq 2^{n+1}$. Thus when is $n+1 \leq \sqrt[10]{2}n$? Sufficient is: $1+\frac{1}{n} \leq \sqrt[10]{2}$. And that happens when: $\frac{1}{n} \leq \sqrt[10]{2}-1$. Thus we want: $n \geq \frac{1}{\sqrt[10]{2}-1} \approx 14$. But $n \geq 100$, so we're fine.

Exercise 3. Let f and g be functions that take non-negative values, and suppose $f \in O(g)$. Show that $g \in \Omega(f)$.

Answer Suppose $f \in O(g)$. Then there is a $c \in \mathbb{R}_{>0}$ and a $N \in \mathbb{N}$ such that for all $n \geq N$ we have $0 \leq f(n) \leq cg(n)$. But then also $0 \leq \frac{1}{c}f(n) \leq g(n)$. Since $\frac{1}{c} \in \mathbb{R}_{>0}$ we have exactly $g \in \Omega(f)$.

Exercise 4. (*) Suppose you have algorithms with the six running times listed below. Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most one for of computation. For each of the algorithm, what is the largest input size n for which you would be able to get the result in one hour?

(i) n^2. (ii) n^3. (iii) $100n^2$. (iv) $n\log n$. (v) 2^n. (vi) 2^{2^n}.

Answers

(i) We want to find the largest n such that $n^2 \leq 10^{10}60^2$. Thus $n = 10^5 60 = 6000000$.

(ii) $n^3 \leq 10^{10}60^2$. Thus $n \leq 10^{\frac{10}{3}}60^{\frac{2}{3}} \approx 33019.2$. Thus $n = 33019$.

(iii) Dividing the answer from (i) by $\sqrt{100} = 10$ yields $n = 600000$.

(iv) By trying, we find: $n = 2889069821505$.

(v) $2^n \leq 10^{10}60^2$ thus $n \leq \log_2 10^{10}60^2 \approx 45.03$. Thus $n = 45$.

(vi) $2^{2^n} \leq 10^{10}60^2$ thus $n \leq \log_2\log_2 10^{10}60^2 \approx 5.49$. Hence $n = 5$.

Exercise 5.

(i) Let Σ be a finite alphabet whose elements are ordered. For $v, w \in \Sigma^*$ (words over Σ), there is an algorithm to determine whether $v <^* w$, where $<^*$ denotes the lexicographic order (dictionary like ordering). Determine the complexity of the algorithm.

(ii) Repeat for $v, w \in \mathbb{N}^*$, this time the complexity in terms of the length of the list and it's largest element.

Answers

(i) The algorithm to compare two words $v, w \in \Sigma^*$ is the following.

```
function COMPARE($v, w$)
    for $i$ from 1 to $\min\{\text{length}\, v, \text{length}\, w\}$ do
        if $v < w$ then
            return yes
        if $v > w$ then
            return no
    return yes
```

Note that each iteration of the loop uses a constant amount of time. In the best-case ($v_1 < w_1$) the algorithm uses one iteration and thus runs in constant time. In the worst-case it runs in $O(n)$ where $n = \min\{\text{length}\, v, \text{length}\, w\}$.

- In this case, each iteration of the loop uses an amount of time dependant on the size of v_i and w_i. (One frequently sees a $n \in \mathbb{N}$ as a word over the alphabet $\{0, 1, \ldots, 9\}$.) Still, in the best case, $v_1 = 0 < 1 = w_1$, the algorithm runs in constant time. In general the algorithm certainly runs in $O(nm)$, where $n = \min\{\text{length}\, v, \text{length}\, w\}$ and $m = \max\{\min\{v_i, w_i\}; 1 \leq i \leq n\}$.

Exercise 6. (†)

The Towers of Hanoi is an ancient puzzle, originating with Hindu priests in the great temple of Benares. Suppose we are given three towers, or more humbly, three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape (see image above). The objective of the puzzle is to move the entire stack to another rod (say from A to B), obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

If there are three discs in A the puzzle can be solved as follows: move A to B ; move A to C; move B to C; move A to B; move C to A; move C to B; move A to B. Hence, in 7 steps.

The inventors of the puzzle stated that the world will end if you solve the above puzzle with 64 rings!

Write a procedure/algorithm that solves the above puzzle and give an estimate on the number of moves one needs to solve the Hanoi puzzle.

Answer The solution is recursive. To solve the puzzle for $n = 1$ is easy: just move A to B. If we can solve the puzzle for $n - 1$, then we can solve it for n. Without loss of generality, assume we want to move a stack of discs on A to B. Move the first $n - 1$ discs to C. Then move the remaining disc on A to B. Finally, move the $n - 1$ discs on C to B. In pseudocode:

```
function F($n, s, t$)
    if $n = 1$ then
        Move $s$ to $t$
    else
        $\{u\} \leftarrow \{\text{A}, \text{B}, \text{C}\} - \{s, t\}$
        $F(n-1, s, u)$
        Move $s$ to $t$
        $F(n-1, u, t)$
```

Define $T(n)$ to be the number of moves required to solve the puzzle with n discs. Then $T(1) = 1$ and $T(n+1) = 2T(n) + 1$. Lets try to expand this:

$$\begin{aligned} T(n) &= 2^1 T(n-1) + 2^0 \\ &= 2^2 T(n-2) + 2^1 + 2^0 \\ &\vdots \\ &= 2^{n-1} T(1) + \cdots + 2^0 \\ &= 2^{n-1} + \cdots + 2^0 \end{aligned}$$

Define $S = 2^{n-1} + \cdots + 2^0$. Then $2S = 2^n + \cdots + 2^1$ and $2S - S = 2^n - 2^0$. Thus $S = T(n) = 2^n - 1$. Thus for $n = 64$ you need 18446744073709551615 moves!