Exercise Sheet 2
COMS10017 Algorithms 2020/2021

Reminder: $\log n$ denotes the binary logarithm, i.e., $\log n = \log_2 n$.

1 Θ and Ω

1. Prove that the following two statements are equivalent:

   (a) $f \in \Theta(g)$.

   (b) $f \in O(g)$ and $g \in O(f)$.

   Solution. In order to prove that two statements are equivalent, we assume first that the first statement holds and we deduce that the second statement then holds as well. Then we assume that the second statement holds and we deduce that the first statement then holds as well.

   Let's first assume that $f \in \Theta(g)$. This means that there are constants c_1, c_2, n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$, for every $n \geq n_0$.

   To show that $f \in O(g)$, we need to show that there are constants c, n_0' such that $f(n) \leq cg(n)$, for every $n \geq n_0'$. This follows immediately by choosing $c = c_2$ and $n_0' = n_0$ as above.

   To show that $g \in O(f)$, we need to show that there are constants c, n_0' such that $g(n) \leq cf(n)$, for every $n \geq n_0'$. This follows immediately by choosing $c = \frac{1}{c_1}$ and $n \geq n_0'$.

   Next, we assume that $f \in O(g)$ and $g \in O(f)$. This implies that there are constants c_1, n_1 such that $f(n) \leq c_1 g(n)$, for every $n \geq n_1$, and constants c_2, n_2 such that $g(n) \leq c_2 f(n)$, for every $n \geq n_2$. We need to show that there are constants d_1, d_2, n_0 such that $d_1 g(n) \leq f(n) \leq d_2 g(n)$, for every $n \geq n_0$. We can chose $d_2 = c_1$, $d_1 = \frac{1}{c_2}$, and $n_0 \geq \max\{n_1, n_2\}$. ✓

2. Prove that the following two statements are equivalent:

   (a) $f \in \Omega(g)$.

   (b) $g \in O(f)$.

   Solution. Let's first assume that $f \in \Omega(g)$. This means that there are constants c_1, n_1 such that $c_1 g(n) \leq f(n)$, for every $n \geq n_1$. We need to show that there are constants c_2, n_2 such that $g(n) \leq c_2 f(n)$, for every $n \geq n_2$. We can pick $c_2 = \frac{1}{c_1}$ and $n_2 = n_1$.

   The reverse direction, i.e., assuming that $g \in O(f)$ and deducing that $f \in \Omega(g)$ is very similar. ✓

3. Let $c > 1$ be a constant. Prove or disprove the following statements:

   (a) $\log_c n \in \Theta(\log n)$.

   Solution. Recall the definition of Θ: A function $f(n) \in \Theta(g(n))$ if there are constants c_1, c_2, n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for every $n \geq n_0$. Hence, we need to find constants c_1, c_2, n_0 such that

   $$c_1 \log n \leq \log_c n \leq c_2 \log n \ ,$$

   for every $n \geq n_0$. Observe that $\log_c n = \frac{\log n}{\log c}$. We can hence chose $c_1 = c_2 = \frac{1}{\log c}$ and $n_0 = 1$, since $c_1 \cdot \log n = c_2 \cdot \log n = \log_c n$. This clearly holds for every $n \geq 1$. ✓

   (b) $\log(n^c) \in \Theta(\log n)$.

   Solution. Again, we need to find constants c_1, c_2, n_0 such that

   $$c_1 \log n \leq \log(n^c) \leq c_2 \log n \ ,$$

   for every $n \geq n_0$. Observe that $\log(n^c) = c \log n$. We can hence chose $c_1 = c_2 = c$ and $n_0 = 1$. ✓

4. Let $c > 2$ be a constant. Prove or disprove the following statement:

   $$2^n \in \Theta(c^n) \ .$$

   Solution. This statement is wrong. We will show that $c^n \notin O(2^n)$. This disproves this statement since if $f \in \Theta(g)$ then $g \in O(f)$ as well.

   For the sake of a contradiction, suppose that $c^n \in O(2^n)$. Then there are constants d, n_0 such that

   $$c^n \leq d \cdot 2^n \ ,$$

   for every $n \geq n_0$. Taking logarithms on both sides, we obtain the equivalent inequality:

   $$\begin{aligned} n \log(c) &\leq \log(d2^n) = \log(d) + n \\ n &\leq \frac{\log(d)}{\log(c) - 1} \ . \end{aligned}$$

   Observe that we only obtain the last inequality since $c > 2$ (since $c > 2$ we also have $\log c > 1$ and $\log(c) - 1 > 0$). This inequality hence does not hold for every $n > \frac{\log(d)}{\log(c)-1}$. This is a contradiction to the assumption that it holds for every $n \geq n_0$. ✓

2 O-notation

1. Consider the following functions:

   $$f_1 = 2^{\sqrt{n}}, f_2 = \log^2(20n), f_3 = n!, f_4 = \frac{1}{2}n^2/\log(n), f_5 = 4\log^2(n), f_6 = 2^{\sqrt{\log n}} \ .$$

   Relabel the functions such that $f_i \in O(f_{i+1})$ (no need to give any proofs here).

Solution.

$$O(\log^2(20n)) \subseteq O(4\log^2(n)) \subseteq O(2^{\sqrt{\log n}}) \subseteq O(\frac{1}{2}n^2/\log(n)) \subseteq O(2^{\sqrt{n}}) \subseteq O(n!)$$

Observe that $\log^2(20n) = \Theta(4\log^2(n))$. We could therefore also swap the positions of the first two functions. ✓

2. Give functions f, g such that $f(n) \in O(g(n))$ and $2^{f(n)} \notin O(2^{g(n)})$.

Solution. Consider for example $f(n) = \log n$ and $g(n) = \frac{1}{2}\log n$. Then clearly $f(n) = O(g(n))$, but $2^{f(n)} \notin 2^{g(n)}$, since $2^{f(n)} = 2^{\log n} = n$ and $2^{g(n)} = 2^{\frac{1}{2}\log n} = n^{\frac{1}{2}} = \sqrt{n}$, and $n \notin O(\sqrt{n})$. ✓

3 Runtime Analysis

Algorithm 1

Require: Int $n \geq 1$
  $x \leftarrow 0$
  for $i = 1 \dots n$ **do**
    for $j = 1 \dots n$ **do**
      $x \leftarrow x + i \cdot j$
    end for
  end for
  return x

Algorithm 2

Require: Int $n \geq 1$
  $x \leftarrow 0$
  for $i = 1 \dots n$ **do**
    for $j = i \dots n$ **do**
      $x \leftarrow x + i \cdot j$
    end for
  end for
  return x

Algorithm 3

Require: Int $n \geq 1$
  $x \leftarrow 0$
  $i \leftarrow 1$
  while $i \leq n$ **do**
    for $j = 1 \dots n$ **do**
      $x \leftarrow x + i \cdot j$
    end for
    $i \leftarrow 2 \cdot i$
  end while
  return x

Algorithm 4

Require: Int $n \geq 1$
  $x \leftarrow 0$
  $i \leftarrow 1$
  while $i \leq n$ **do**
    for $j = 1 \dots i$ **do**
      $x \leftarrow x + i \cdot j$
    end for
    $i \leftarrow 2 \cdot i$
  end while
  return x

Determine the runtimes of Algorithms 1,2,3 and 4 using Big "Theta" notation.

Solution.

1. Algorithm 1 runs in time $\Theta(n^2)$.

2. Algorithm 2 runs in time $\Theta(n^2)$:

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i}^{n}\Theta(1) &= \Theta\left(\sum_{i=1}^{n}\sum_{j=i}^{n}1\right) = \Theta\left(\sum_{i=1}^{n} n - i + 1\right) = \Theta\left(\sum_{i=1}^{n}(n+1) - \sum_{i=1}^{n} i\right) \\ &= \Theta\left(n(n+1) - \frac{n(n+1)}{2}\right) = \Theta\left(\frac{n(n+1)}{2}\right) = \Theta(n^2)\ .\end{aligned}$$

3. Observe that the inner loop in Algorithm 3 always requires $\Theta(n)$ time in total. It remains to determine how often the outer loop is executed. To this end, for $j \geq 1$, let i_j be the value of i at the beginning of iteration j. Then, $i_1 = 1$, and $i_j = 2 \cdot i_{j-1} = 4 \cdot i_{j-2} = \dots = 2^{j-1}$. Let k be the last iteration of the loop. Then, $2^{k-1} \leq n$ and $2^k > n$. We have $2^k > n \Rightarrow k > \log n$, and similarly we get $k - 1 \leq \log n$, which implies $k \leq \log(n) + 1$. We thus have the conditions: $\log n < k < \log(n) + 1$, and since k is an integer, we obtain $k = \lfloor \log(n) + 1 \rfloor$. Hence, the outer loop is executed $\lfloor \log(n) + 1 \rfloor$ times. The total runtime is therefore $\Theta(n \log n)$.

4. Observe that in Algorithm 4 the inner loop runs in time $\Theta(i)$. As demonstrated in the previous exercise, the outer loop is executed $\lfloor \log(n) + 1 \rfloor$ times and the variable i takes on values $1, 2, 4, 8, \ldots, 2^{\lfloor \log(n)+1 \rfloor - 1}$. We can thus bound the runtime as follows:

$$\Theta\left(\sum_{j=1}^{\lfloor \log(n)+1 \rfloor - 1} 2^j\right) = \Theta(2^{\lfloor \log(n)+1 \rfloor}) = \Theta(n).$$

✓

4 Optional and Difficult Questions

Exercises in this section are intentionally more difficult and are there to challenge yourself.

4.1 Average Case Runtime of Linear Search

For integers $k, n \geq 1$ let $S_k(n)$ be the set of all integer arrays of length n where every array entry is taken from the set $\{0, 1, 2, \ldots, k-1\}$.

1. What is the average case runtime of linear search on $S_3(n)$?
2. What is the average case runtime of linear search on $S_C(n)$, for any constant C?
3. What is the average case runtime of linear search on $S_n(n)$?
4. What is the average case runtime of linear search on $S_{\sqrt{n}}(n)$?

Solution. We will give an analysis for a general k below.

Similar to how we proceeded in the lecture, we will bound the average position of the left-most 0. Observe that considering 0 instead of any of the other values $1, 2, \ldots, k-1$ as the target value can be done without loss of generality due to symmetry of the problem. The position of the left-most 0 determines the runtime in Big-O notation since the algorithm stops as soon as a 0 is found.

This quantity can be computed as follows (observe that $|S_k(n)| = k^n$):

$$\begin{aligned} AVG &= \frac{1}{|S_k(n)|}\left(\left(\sum_{i=1}^{n} i \cdot (k-1)^{i-1} k^{n-i}\right) + (n+1)(k-1)^n\right) \\ &= \left(\sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1} k^{n-i}}{k^n}\right) + (n+1)\frac{(k-1)^n}{k^n} \\ &= \underbrace{\left(\sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1}}{k^i}\right)}_{I} + (n+1)\left(\frac{k-1}{k}\right)^n, \end{aligned} \tag{1}$$

where we accounted a runtime of $n+1$ if the input does not contain any 0s.

We will now focus on bounding the term I by expanding the two terms I and $\frac{k-1}{k} \cdot I$:

$$\begin{aligned}
\sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1}}{k^i} &= 1 \cdot \frac{1}{k} + 2 \cdot \frac{k-1}{k^2} + 3 \cdot \frac{(k-1)^2}{k^3} + \cdots + n \cdot \frac{(k-1)^{n-1}}{k^n} \ , \\
\frac{k-1}{k} \cdot \sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1}}{k^i} &= 1 \cdot \frac{k-1}{k^2} + 2 \cdot \frac{(k-1)^2}{k^3} + \cdots + (n-1) \cdot \frac{(k-1)^{n-1}}{k^n} + n \cdot \frac{(k-1)^n}{k^{n+1}} \ .
\end{aligned}$$

We next compute the first line "minus" the second and obtain:

$$\begin{aligned}
\frac{1}{k} \sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1}}{k^i} &= \frac{1}{k} + \frac{k-1}{k^2} + \frac{(k-1)^2}{k^3} + \cdots + \frac{(k-1)^{n-1}}{k^n} + n \cdot \frac{(k-1)^n}{k^{n+1}} \\
&= \left(\frac{1}{k} \cdot \sum_{i=0}^{n-1} \left(\frac{k-1}{k} \right)^i \right) + n \cdot \frac{(k-1)^n}{k^{n+1}} \\
&= \frac{1}{k} \cdot \frac{1 - (\frac{k-1}{k})^n}{1 - \frac{k-1}{k}} + n \cdot \frac{(k-1)^n}{k^{n+1}} \\
&\leq \frac{1}{k} \cdot \frac{1}{\frac{1}{k}} + n \cdot \frac{(k-1)^n}{k^{n+1}} = 1 + n \cdot \frac{(k-1)^n}{k^{n+1}} \ ,
\end{aligned}$$

where we used the formula $\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r} \leq \frac{1}{1-r}$ for computing a geometric series. We thus obtained:

$$\sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1}}{k^i} \leq k + k \cdot n \cdot \frac{(k-1)^n}{k^{n+1}} \ .$$

Next, we plug this bound into Inequality 1, which yields

$$\begin{aligned}
AVG &= \left(\sum_{i=1}^{n} i \cdot \frac{(k-1)^{i-1}}{k^i} \right) + (n+1) \left(\frac{k-1}{k} \right)^n \\
&\leq k + k \cdot n \cdot \frac{(k-1)^n}{k^{n+1}} + (n+1) \cdot \left(\frac{k-1}{k} \right)^n \\
&= k + n \cdot \left(\frac{k-1}{k} \right)^n + (n+1) \left(\frac{k-1}{k} \right)^n \\
&= k + (2n+1) \left(\frac{k-1}{k} \right)^n \ .
\end{aligned}$$

Last, we will argue that $(2n+1) \cdot \left(\frac{k-1}{k}\right)^n$ is small. To this end, we will use the inequality $1 + x \leq e^x$, which holds for every x, and we obtain

$$\begin{aligned}
(2n+1) \cdot \left(\frac{k-1}{k} \right)^n &\leq (2n+1) \cdot \left(1 - \frac{1}{k} \right)^n \\
&\leq (2n+1) \cdot \left(e^{-\frac{1}{k}} \right)^n \\
&= \frac{2n+1}{e^{\frac{n}{k}}} \ .
\end{aligned}$$

Last, observe that $e^{\frac{n}{k}} \geq \frac{n}{k}$ holds for every value of n and k. Hence,

$$\frac{2n+1}{e^{\frac{n}{k}}} \leq \frac{2n+1}{\frac{n}{k}} = k \cdot \frac{2n+1}{n} = k \cdot (2 + \frac{1}{n}) \leq 2k+1 \ .$$

We thus obtain $AVG \leq k + 2k + 1 = 3k + 1 = \Theta(k)$. ✓