FTP_Alg_Week 1: Exercises (with solutions)

jungkyu.canci@hslu.ch

HS2024

Exercise 1 *A sorting method with "Big-Oh" complexity $O(n \log n)$ spends exactly 1 millisecond to sort $1'000$ data items. Assuming that time $T(n)$ of sorting n items is directly proportional to $n \log n$, that is, $T(n) = c \cdot n \log n$, derive a formula for $T(n)$, given the time $T(N)$ for sorting N items, and estimate how long this method will sort $1'000'000$ items.*

Solution: We know that $T(1'000) = 1\text{ms}$, thus $c \cdot 1'000 \log 1'000 = 1\text{ms}$ therefore

$$c = \frac{1ms}{1'000 \log 1'000}$$

(no need to calculate the value of c). Therefore

$$\begin{aligned}
T(1'000'000) &= c \cdot 1'000'000 \cdot \log 1'000'000 \\
&= \frac{\log 1'000'000}{1'000 \log 1'000} 1'000'000\text{ms} \\
&= \frac{\log(1'000^2)}{\log 1'000} 1'000\text{ms} \\
&= \frac{2 \log 1'000}{\log 1'000}\text{s} = 2s.
\end{aligned}$$

Note that it does not matter the argument of the logarithmus.

Exercise 2 *A quadratic algorithm with processing time $T(n) = cn^2$ spends $T(N)$ seconds for processing N data items. How much time will be spent for processing $n = 5000$ data items, assuming that $N = 100$ and $T(N) = 1\text{ms}$?*

Solution: From $T(100) = 1\text{ms}$ we obtain

$$c = \frac{1\text{ms}}{100^2}$$

Thus

$$T(5000) = c \cdot (5000)^2 = \frac{1\text{ms}}{100^2} \cdot 50^2 \cdot 100^2 = 2500\text{ms} = 2.5\text{s}.$$

Exercise 3 *An algorithm with time complexity $O(f(n))$ and processing time $T(n) = c \cdot f(n)$, where $f(n)$ is a known function of n, spends 10 seconds to process 1'000 data items. How much time will be spent to process 100'000 data items if $f(n) = n$ and $f(n) = n^3$?*

Solution: We have

$$c = \frac{10\text{s}}{f(1'000)}$$

Thus

$$T(100'000) = \frac{10\text{s}}{f(1'000)} \cdot f(100'000) = \frac{10\text{s}}{f(10^3)} \cdot f(10^5)$$

Case $f(n) = n$.

$$T(100'000) = \frac{10\text{s}}{10^3} \cdot 10^5 = 1'000\text{s}.$$

Case $f(n) = n^3$.

$$T(100'000) = \frac{10\text{s}}{(10^3)^3} \cdot (10^5)^3 = \frac{10\text{s}}{10^9} \cdot 10^{15} = 10^7\text{s} = 10'000'000\text{s}.$$

Note that 10^3 second are about 17 minutes, 10^7 seconds are about 116 days.

Exercise 4 *Assume that each of the expressions below gives the processing time $T(n)$ spent by an algorithm for solving a problem of size n. Select the dominant term(s) having the steepest increase in n and specify the lowest Big-Oh complexity of each algorithm.*

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

Solution:

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n\log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n,\ n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$	$O(\log n)$

Exercise 5 *The statements below show some features of "Big-Oh" notation for the functions $f \equiv f(n)$ and $g \equiv g(n)$. Determine whether each statement is TRUE or FALSE and correct the formula in the latter case.*

Statement	Is it TRUE or FALSE?	If it is FALSE then write the correct formula
Rule of sums: $O(f+g) = O(f) + O(g)$		
Rule of products: $O(f \cdot g) = O(f) \cdot O(g)$		
Transitivity: if $g = O(f)$ and $h = O(f)$ then $g = O(h)$		
$5n + 8n^2 + 100n^3 = O(n^4)$		
$5n+8n^2+100n^3 = O(n^2 \log n)$		

Solution:

Statement	Is it TRUE or FALSE?	If it is FALSE then write the correct formula
Rule of sums: $O(f+g) = O(f) + O(g)$	FALSE	$O(f+g) =$ $\max\{O(f), O(g)\}$
Rule of products: $O(f \cdot g) = O(f) \cdot O(g)$	TRUE	
Transitivity: if $g = O(f)$ and $h = O(f)$ then $g = O(h)$	FALSE	if $g = O(f)$ and $f = O(h)$ then $g = O(h)$
$5n + 8n^2 + 100n^3 = O(n^4)$	TRUE	
$5n + 8n^2 + 100n^3 = O(n^2 \log n)$	FALSE	$5n + 8n^2 + 100n^3 = O(n^3)$

Exercise 6 *Work out the computational complexity of the following piece of code:*

```
Algorithm
  for i = n to 1 do
    for j = 1 to n do
      for k = 0 to n do
        ... // constant number of operations
        k = k + 2
      end for
      j = j * 2
    end for
    i = i / 2
  end for
```

Solution: They are three nested loops, loop 1 that contains loop 2, which contains loop 3. In loop 1 the variable i keeps halving so it goes round $\log_2 n$ times. For each i, loop 2 goes round $\log_2 n$ times, because the variable j keeps doubling. Loop 3 goes round $n/2$ times, because k assume all even number smaller or equal than n. Since the loops are nested the total running time is

$$O\left(\frac{n}{2}(\log_2 n)(\log_2 n)\right) = O\left(n(\log_2 n)^2\right).$$

Exercise 7 (*) *Work out the computational complexity of the following piece of code:*

```
sum=0
for i = 1 to n do
  for j = n to 1 do
    for k = j to n do
      sum=sum + (i+j*k)
      k = k + 2
    end for
    j = j / 2
  end for
  i = i * 2
end for
```

Solution: The loop 1 (starting from outside) goes round $\log_2 n$ repetitions. Loop 2 we goes round $\log_2(n+1)$, which is $O(\log n)$ (more precisely $\Theta(\log n)$, since

$$\log_2(n+1) = \log_2\left(n \cdot \frac{n+1}{n}\right) \leq \log_2 2 + \log_2 n\cdot) = \log_2(2n\cdot)1 + \log_2(n) \in O(\log n)$$

We have used that $\frac{n+1}{n} \leq 2$ for all $n \in \mathbb{N}$. Furthermore note that for the transformation formula of base change for logarithm

$$\log_b n = \frac{1}{\log_a b} \cdot \log_a n$$

does not matter the base of the logarithm in the expression of $O(\log n)$. Loop goes $(n-j)/2+1$ times the operation

$$\mathrm{sum} = \mathrm{sum} + (\mathrm{i} + \mathrm{j} * \mathrm{k})$$

whioch takes constant time. Thus the inner loop goes $O(n)$. Therefore the running time is in $O\left(n(\log_2 n)^2\right)$.

Note that before we considered the upper bound for $O(n)$ for the running times that the inner loop. We could be subtler in the analysis by considering the exact running time in the middle and inner loop. Let us assume $n = 2^k$. The index j of the middle assumes the values

$$2^k, \frac{2^k}{2} = 2^{k-1}, \ldots, 2^r =, \ldots, \frac{n}{2^k} = 1, 0.$$

With $j = n$ we have 1 iteration. With $j = 2^{k-1}$ we have $(2^k - 2^{k-1})/2 + 1$ iterations. With $j = 2^r$, we have $(2^k - 2^r)/2 + 1$...and so on. With $j = 1$ we have about $(2^k - 1)/2 + 1$ iterations. With $j = 0$ we have $(2^k)/2$ iterations. Let c be running time of the sum in loop 3, therefore the running time of the inner and middle loops is c times

$$1 + ((2^k - 2^{k-1})/2 + 1) + \ldots + ((2^k - 2^r)/2 + 1) + \ldots + ((2^k - 1)/2 + 1) + (2^k)/2 =$$
$$= 1 + k \cdot 2^{k-1} - (1 + 2 + \ldots + 2^{k-1})/2 + k = 1 + k \cdot 2^{k-1} - (2^k - 1)/2 =$$
$$= 3/2 + k + (k-1)2^{k-1} = 3/2 + \log_2 n + (\log_2 n - 1)n/2 \in \Theta(n \log n)$$

Thus the running time is $\Theta(\log n) \cdot \Theta(n \log n) = \Theta(n \log^2 n)$.

Exercise 8 (*) *Running time $T(n)$ of processing n data items with a given algorithm is described by the recurrence: $T(n) = k \cdot T\left(\frac{n}{k}\right) + c \cdot n;\quad T(1) = 0$. Derive a closed form formula for $T(n)$ in terms of c, n, and k. What is the computational complexity of this algorithm in a "Big-Oh" sense?* [Hint: To have the well-defined recurrence, assume that $n = k^m$ with the integer $m = \log_k n$ and k.]

Solution 1 (using the hint): We assume that $n = k^m$. For any integer $1 \le r \le m$ we have

$$T(k^r) = k \cdot T(k^{r-1}) + c \cdot k^r$$

Thus we have

$$\begin{array}{lcl} T(k) & = & k \cdot T(1) + k \cdot c = k(T(1) + c) \\ T(k^2) & = & k \cdot T(k) + c \cdot k^2 = k^2(T(1) + c) + c \cdot k^2 = k^2(T(1) + 2c) \\ \vdots & \vdots & \vdots \\ T(k^m) & = & k \cdot T(k^{m-1}) + c \cdot k^m \\ & = & k^m(T(1) + (m-1) \cdot c) + c \cdot k^m = k^m(T(1) + m \cdot c) \end{array}$$

Since $m = \log_k k^m = \log_k n$ (because $n = k^m$). We have proven that

$$T(n) \in O(n(T(1) + \log_k n)) = O(n \log n).$$

Remarks to Solution 1: In the last displaymath in Solution 1 we have used the first property listed in solutions of Exercise 5:

$$O(f + g) = \max\{O(f), O(g)\}.$$

In the second last dispalymath of Solution 1 (list of $T(k), \ldots, T(k^m)$), we have actually proven by induction that the identity

$$T(k^r) = k^r(T(1) + r \cdot c)$$

holds for all positive integer r.
Recall that so called **mathematical induction principle**. Let $p(n)$ a statement that involve an arbitrary non negative integer n. Let n_0 be a given non negative integer. Suppose that the statement $p(n_0)$ is true and by assuming that $p(s-1)$ is true for an arbitrary positive integer s, then we are able to prove (with logical deductions) that $p(s)$ is true, then the property holds for all non negative integers $n \ge n_0$.
Sometimes it is useful to use the equivalent form: **mathematical induction principle (second form)**. Let $p(n)$ a statement that involve an arbitrary non negative integer n. Let n_0 be a given non negative integer and s an arbitrary integer greater than n_0. Suppose that the statement $p(n_0)$ and by assuming that $p(i)$ is true for all$n_0 \le i \le s-1$, then we are able to prove (with logical deductions) that $p(s)$ is true, then the property holds for all non negative integers $n \ge n_0$.

One can prove by induction (with no "telscoping") that $T(n) \in O(n \log n)$.
Solution 2: One can use **Master Theorem** (see Chap 1_3) case 2.