Australian National University
Research School of Computer Science

Computer Science COMP3600/COMP6466
Answers to Tutorial One (2016)

In the following we provide some reference answers to the problems of Tutorial 1. The relevant sections of the textbook should be consulted in the first place in order to get more information.

Question 1.

(i) Review the definitions of the basic notations such as Θ, O, Ω, o, ω.

See the slides from Lecture 2.

(ii) Rank the following functions in the order of growth:
$n^2 \lg n$, $n^{3/2} \lg^4 n$, $3^{0.5n}$, $\lg^* n \lg^2 n$, $n^{1.44}$, $\sqrt{n^{1/3} \lg n}$, $n^{\lg \lg n}$, $\sqrt{n}$, n^5.

$\lg^* n \lg^2 n$, $\sqrt{n^{1/3} \lg n}$, $\sqrt{n}$, $n^{1.44}$, $n^{3/2} \lg^4 n$, $n^2 \lg n$, n^5, $n^{\lg \lg n}$, $3^{0.5n}$

The most important principles are that logarithmic functions grow slower than powers, and powers grow slower than exponential functions.

To compare the relative order of $n^{\lg \lg n}$ and $3^{0.5n}$, take their logarithms.

Question 2.

Review the various techniques for bounding summations.

See notes from Lecture 4.

Estimate the following functions, using $\Theta(\,)$ notation:

$$\sum_{k=1}^{n} \frac{k^2}{7^k}$$

If the ratio between adjacent terms is greater than or equal to a constant greater than 1, or less than or equal to a constant less than 1, the series can be bounded by a geometric sum, which is Θ(largest term). In this case,

$$\frac{a_{k+1}}{a_k} = \frac{(k+1)^2/7^{k+1}}{k^2/7^k} = \frac{1}{7}(\frac{k+1}{k})^2 \leq \frac{4}{7} = r \text{ for } k \geq 1,$$

which gives $\sum_{k=1}^{n} \frac{k^2}{7^k} \leq \sum_{k=1}^{n} a_1 r^{k-1} = \Theta(1)$ (see slides 9 and 10 of Lecture 4 for more details).
For a lower bound, take, say, the first term:

$$\sum_{k=1}^{n} \frac{k^2}{7^k} \geq a_1 = 1/7 = \Theta(1).$$

The two bounds together imply $\sum_{k=1}^{n} \frac{k^2}{7^k} = \Theta(1)$.

$$\sum_{k=2}^{n} \frac{k^{1/2}}{\lg k}$$

In the case of slowly-changing terms, such as terms of polynomial size, bound by the maximum and minimum terms. For an upper bound, take the number of terms times the maximum term:

$$\sum_{k=2}^{n} \frac{k^{1/2}}{\lg k} \leq n \times \frac{n^{1/2}}{\lg n} = \frac{n^{3/2}}{\lg n}.$$

For a lower bound, take half of the number of terms times the minimum term from the larger half of the terms. We will use terms $k = \lceil n/2 \rceil \ldots n$:

$$\sum_{k=2}^{n} \frac{k^{1/2}}{\lg k} \geq \frac{n}{2} \times \frac{(n/2)^{1/2}}{\lg(n/2)} \geq \frac{1}{4} \frac{n^{3/2}}{\lg n}.$$

So the value of the sum is $\Theta(n^{3/2}/\lg n)$. For more details, see slides 7 and 8 of Lecture 4.

$$\sum_{k=1}^{n} k^{7/2}$$

$\sum_{k=1}^{n} k^{7/2} \leq \sum_{k=1}^{n} n^{7/2} = n^{9/2}$, thus, $\sum_{k=1}^{n} k^{7/2} = O(n^{9/2})$.
Also, $\sum_{k=1}^{n} k^{7/2} \geq \sum_{k=\lceil n/2 \rceil}^{n} k^{7/2} \geq \frac{n}{2}(\frac{n}{2})^{7/2} = (\frac{n}{2})^{9/2}$, i.e, $\sum_{k=1}^{n} k^{7/2} = \Omega(n^{9/2})$.
Thus,

$$\sum_{k=1}^{n} k^{7/2} = \Theta(n^{9/2}).$$

Question 3. Review the two approaches for solving recurrences, and apply them to solve the following recurrences.

See notes for Lecture 5.

- (a) $T(n) = T(n/3) + n^{5/4}$.
  We apply the iteration method. Note that $k = \log_3 n$ when $\frac{n}{3^k} = 1$. Let's assume that $T(n)$ is defined for all rational numbers n, and that $T(1)$ is a constant. Then,

$$\begin{aligned} T(n) &= T(n/3) + n^{5/4} \\ &= T(n/3^2) + n^{5/4} + (n/3)^{5/4} \\ &= \cdots \\ &= T(1) + n^{5/4} + (n/3)^{5/4} + (n/3^2)^{5/4} + \ldots + (n/3^k)^{5/4}. \end{aligned}$$

  Apart from $T(1)$, this is a series of geometric type so its sum is $\Theta(n^{5/4})$ (now we consider $T(n)$ only for natural numbers n). The term $T(1)$ is negligible in comparison, the answer thus is $T(n) = \Theta(n^{5/4})$.

- (b) $T(n) = 3T(n/2) + 4n$.

  Apply the iteration method again. Here, we also assume that $T(n)$ is defined for all rational numbers n (but later, when determining asymptotic bounds, we only consider $T(n)$ for natural numbers). This time $k = \log_2 n$ steps are needed.

$$\begin{aligned} T(n) &= 3T(n/2) + 4n \\ &= 3^2T(n/2^2) + 3 \cdot 4 \cdot n/2 + 4n \\ &= 3^3T(n/2^3) + (3/2)^2 4n + (3/2)4n + 4n \\ &= \cdots \\ &= 3^kT(1) + (3/2)^{k-1}4n + \cdots + (3/2)4n + 4n. \end{aligned}$$

  The series, apart from the term $3^kT(1)$, is of geometric type so its sum is $\Theta(\text{largest term}) = \Theta((3/2)^{k-1}4n) = \Theta((3/2)^{\log_2 n - 1}4n) = \Theta(3^{\log_2 n}) = \Theta(n^{\log_2 3})$. Furthermore, $3^kT(1) = 3^{\log_2 n}T(1) = n^{\log_2 3}T(1) = \Theta(n^{\log_2 3})$, so the answer is $T(n) = \Theta(n^{\log_2 3})$.

- (c) $T(n) = T(\lfloor\sqrt{n}\rfloor) + n \log n$.

  We use the substitution method (i.e., mathematical induction), starting with the guess that the answer might be $T(n) = \Theta(n \lg n)$. The asymptotic lower bound $T(n) = \Omega(n \lg n)$ follows immediately from the form of the recurrence, so we only need to prove $T(n) = O(n \lg n)$. To avoid having either n or $\lfloor\sqrt{n}\rfloor$ equal to 1 (which has a logarithm of 0), we start the induction at $n = 4$, for which we know that the result holds because $T(4)$ is assumed to be a constant. (This is the base case.)

Assume that for some positive constant c (the same that was used in the base case), we have

$$T(n') \leq cn' \log n'$$

for all n' such that $4 \leq n' < n$. (This is the induction hypothesis.) Then, applying the recurrence, we have

$$\begin{aligned} T(n) &= T(\lfloor \sqrt{n} \rfloor) + n \log n \\ &\leq c\sqrt{n} \log \sqrt{n} + n \log n \\ &\leq \left(\tfrac{1}{2} c n^{-1/2} + 1\right) n \log n \\ &\leq cn \log n, \end{aligned}$$

as long as c is large enough to satisfy $\frac{1}{2} cn^{-1/2} + 1 \leq c$, and as long as the hypothesis can be used to bound $T(\lfloor \sqrt{n} \rfloor)$, that is, $\lfloor \sqrt{n} \rfloor \geq 4$, so $n \geq 16$.
The cases $n = 5, \ldots, 15$ have to be checked separately. But $T(n)$ is assumed to be constant for small values of n, so the statement holds for these values too. Of course we have to be careful to choose a large enough c that works throughout the proof.

By induction, we have $T(n) = \Theta(n \log n)$.

Question 4. Today is your birthday, you are 37 years old! You are a great cybersecurity expert and your company has just been breached by hackers who managed to turn the power off. The backup generator is protected by a 3 digit code. Obviously, you forgot the code. Fortunately, you have written down the following note, anticipating such situations: "It is a number greater then triple your age today, square it, you get 11664"; and the note is two months old.

Since power is out, you can only use a manual algorithm to crack your code. The only algorithm available on you desk is the old Babylonian method to compute the square root of a number.

The Babylonian Method

INPUT: n
OUTPUT: approximation of $\sqrt{n}$

1. Start with an initial value x_0.
2. Let $x_i = \frac{x_{i-1} + \frac{n}{x_{i-1}}}{2}$, $i \in \{1, \ldots, i_{max}\}$

3. return $x_{i_{max}}$

One key property of the Babylonian method is that the approximate solution is always greater or equal to the square root value.
We will first prove this property.

1. Using the fact that $(x_{i-1} - \sqrt{n})^2 \geq 0$, show that $\frac{x_{i-1}+\frac{n}{x_{i-1}}}{2} \geq \sqrt{n},\ \forall x_{i-1} > 0$.

$$
\begin{gathered}
(x_{i-1} - \sqrt{n})^2 \geq 0 \\
\Updownarrow \\
x_{i-1}^2 - 2x_{i-1}\sqrt{n} + n \geq 0 \\
\Updownarrow \\
x_{i-1} - 2\sqrt{n} + \frac{n}{x_{i-1}} \geq 0 \\
\Updownarrow \\
x_{i-1} + \frac{n}{x_{i-1}} \geq 2\sqrt{n} \\
\Updownarrow \\
\frac{x_{i-1} + \frac{n}{x_{i-1}}}{2} \geq \sqrt{n}
\end{gathered}
$$

2. Using the Babylonian method, starting with $x_0 = 100$, find the three digit code for your backup generator.

   The first iteration of the algorithm gives

   $$x_1 = \frac{100 + \frac{11664}{100}}{2} = \frac{216.64}{2} = 108.32$$

   Let x denote the code, since I was 36 when the note was written, we have that $x \geq 36*3 = 108$. Based on the property proved earlier, we also have $x \leq 108.32$, therefore $x = 108$.

Great job! The power is on again, so is your screen!
You computer is now able to communicate with all the servers on your grid.
Your initial goal is to check whether the main server is being accessed by hackers.
You can access a list of the servers that are currently being hacked. The list is sorted in increasing server id number. You have 10 seconds to check if the main server is being hacked.
Your grid contains 24000 servers, and a server id comparison action costs you one millisecond.

1. Write an algorithm that is able to tackle your initial goal.
   Use the Binary search algorithm presented in Lecture 1.

2. Specify its input(s) and output(s).
   INPUT: list of hacked servers, main server id
   OUTPUT: Yes or No.

3. Show that your algorithm will always return a solution within the time limit.
   The worst case number of comparisons is bounded by $\lceil \log_2(24000) \rceil = 15$. In a worst case scenario, the time needed to check if the main server is being hacked is 15 milliseconds, which is well within the time limit.