COMP4500/7500 Advanced Algorithms & Data Structures
Sample Solution to Tutorial Exercise 2 (2018/2)[*]

School of Information Technology and Electrical Engineering, University of Queensland

August 24, 2018

1. (See CLRS Exercise 1.2-2, p14 [3rd], p13 [2nd], CLR Exercise 1.4-1, p17 [1st])
Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, assume insertion sort runs in $10n^2$ steps, while merge sort runs in $100n \lg n$ steps, where $\lg n$ stands for $\log_2 n$. For which values of n does insertion sort beat merge sort? You will need to use a bit of algebra, but may then use trial and error if you wish.

   How might one write a sorting algorithm which has the advantages of insertion sort for small inputs and the advantages of merge sort for larger inputs? (For an extension, see CLRS Problem 2-1 p39 [3rd], p37 [2nd], CLR Problem 1-2, p17 [1st]).

   Sample solution. Insertion sort is faster than merge sort when

$$\begin{aligned} & 10n^2 < 100n \lg n \\ \equiv\ & n < 10 \lg n \\ \equiv\ & \frac{n}{10} < \lg n \\ \equiv\ & 2^{\frac{n}{10}} < n \\ \equiv\ & 2 \le n \le 58 \end{aligned}$$

   Observe that for $n = 50$: $2^{\frac{n}{10}} = 2^{\frac{50}{10}} = 2^5 = 32 < 50 = n$,
and for $n = 60$: $2^{\frac{n}{10}} = 2^{\frac{60}{10}} = 2^6 = 64 > 60 = n$.

   Hence $50 < n < 60$. The more exact upper bound on n of 58 can be ascertained by trial and error using a calculator. [Note that the use of asymptotic notation avoids this level of detail, and thus avoids the trial-and-error approach.]

   Merge sort can be rewritten so that it does an insertion sort on inputs of size 58 or less. The modified merge sort now takes fewer steps than the straight merge sort. What is the complexity of this modified merge sort?

2. (CLRS Exercise 3.1-1, p52 [3rd], p50 [2nd], CLR Exercise 2.1-1, p31 [1st])
A function h is asymptotically nonnegative if there exists an n_h such that

$$(\forall n \bullet n \ge n_h \Rightarrow h(n) \ge 0)$$

   Let f and g be asymptotically nonnegative functions. Using the definition of Θ-notation, prove that the function $\max(f(n), g(n)) \in \Theta(f(n) + g(n))$, where for each value of n the function $\max(f(n), g(n))$ returns the maximum of $f(n)$ and $g(n)$. As abbreviations, define functions M and S so that, for all n

$$\begin{aligned} M(n) &= \max(f(n), g(n)) \\ S(n) &= f(n) + g(n) \end{aligned}$$

   Sample solution. Since f and g are asymptotically nonnegative, there exist n_f and n_g such that

$$(\forall n \bullet n \ge n_f \Rightarrow f(n) \ge 0) \text{ and } (\forall n \bullet n \ge n_g \Rightarrow g(n) \ge 0).$$

   Taking n_0 as the maximum of n_f and n_g we find

$$\begin{aligned} & (\forall n \bullet n \ge n_0 \Rightarrow f(n) \ge 0 \wedge g(n) \ge 0) \\ \Rightarrow\ & (\forall n \bullet n \ge n_0 \Rightarrow f(n) + g(n) \ge f(n) \ge 0 \wedge f(n) + g(n) \ge g(n) \ge 0) \\ \equiv\ & (\forall n \bullet n \ge n_0 \Rightarrow S(n) \ge f(n) \ge 0 \wedge S(n) \ge g(n) \ge 0) \\ \Rightarrow\ & (\forall n \bullet n \ge n_0 \Rightarrow S(n) \ge M(n) \ge 0) \\ \Rightarrow\ & M \in O(S) \end{aligned}$$

[*]Copyright © reserved August 24, 2018

Similarly,

$$\begin{array}{cl} & (\forall n \bullet n \geq n_0 \Rightarrow M(n) \geq f(n) \geq 0 \wedge M(n) \geq g(n) \geq 0) \\ \Rightarrow & (\forall n \bullet n \geq n_0 \Rightarrow M(n) + M(n) \geq f(n) + g(n) \geq 0) \\ \equiv & (\forall n \bullet n \geq n_0 \Rightarrow 2 \cdot M(n) \geq S(n) \geq 0) \\ \equiv & (\forall n \bullet n \geq n_0 \Rightarrow M(n) \geq \frac{1}{2} \cdot S(n) \geq 0) \\ \Rightarrow & M \in \Omega(S) \end{array}$$

Finally, $M \in O(S)$ and $M \in \Omega(S)$ together imply $M \in \Theta(S)$.

3. (CLRS Exercise 3.1-3, p53 [3rd], p50 [2nd], CLR Exercise 2.1-3, p31 [1st])
Explain why the statement, "The running time of algorithm A is at least $O(n^2)$", is content-free.

Sample solution. Let the running time of A be $T(n)$. $T(n) \geq O(n^2)$ means (CLRS p47 [3rd], p46 [2nd], p28 [1st]) that $T(n) \geq f(n)$ for some function $f \in O(n^2)$. This is true for any running time $T(n)$, since the function g, such that $g(n) = 0$ for all n, is in $O(n^2)$ and running times are always greater than zero. So the statement tells us nothing about the running time.

4. (CLRS Exercise 3.1-4, p53 [3rd], p50 [2nd], CLR Exercise 2.1-4, p31 [1st])
Is $2^{n+1} \in O(2^n)$? Is $2^{2n} \in O(2^n)$?

Sample solution. $2^{n+1} \in O(2^n)$, but $2^{2n} \notin O(2^n)$.

To show that $2^{n+1} \in O(2^n)$, we must find constants $c, n_0 > 0$ such that: $\forall n \bullet n \geq n_0 \Rightarrow 0 \leq 2^{n+1} \leq c \cdot 2^n$.

Now: $\forall n \bullet n \geq 1 \Rightarrow 0 \leq 2^{n+1} = 2 \cdot 2^n$.

Therefore we can satisfy the definition with $c = 2$ and $n_0 = 1$.

Alternatively, because the two functions are asymptotically non-negative, we can take the limit of their ratios. If the limit is a non-zero constant, the two functions are the same asymptotic complexity.

$$\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} 2 = 2$$

Therefore, $2^{n+1} \in \Theta(2^n)$.

To show that $2^{2n} \notin O(2^n)$ we make use of a proof by contradiction. Assume that $2^{2n} \in O(2^n)$, i.e., there exist constants $c, n_0 > 0$ such that

$$\begin{array}{cl} & \forall n \bullet n \geq n_0 \Rightarrow 0 \leq 2^{2n} \leq c \cdot 2^n \\ \equiv & \forall n \bullet n \geq n_0 \Rightarrow 0 \leq 2^n \cdot 2^n \leq c \cdot 2^n \\ \equiv & \forall n \bullet n \geq n_0 \Rightarrow 0 \leq 2^n \leq c. \end{array}$$

But no constant is bigger than 2^n for all n, so the assumption leads to a contradication and must be false.

Alternatively, because the two functions are asymptotically non-negative, we can take the limit of the ratios of 2^n and 2^{2n}. If the limit is zero then $2^n \in O(2^{2n})$ but $2^n \notin \Theta(2^{2n})$.

$$\lim_{n\to\infty} \frac{2^n}{2^{2n}} = \lim_{n\to\infty} \frac{1}{2^n} = 0$$

Now

$$\begin{array}{cll} & 2^n \notin \Theta(2^{2n}) & \\ \equiv & 2^n \notin O(2^{2n}) \vee 2^n \notin \Omega(2^{2n}) & \text{– property of } \Theta \\ \equiv & 2^n \notin \Omega(2^{2n}) & \text{– as } 2^n \in O(2^{2n}) \\ \equiv & 2^{2n} \notin O(2^n) & \text{– swapping } \Omega \text{ for } O \end{array}$$

5. Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{25}$ of the functions satisfying $g_1 \in \Omega(g_2)$, $g_2 \in \Omega(g_3)$, $\ldots$, $g_{24} \in \Omega(g_{25})$. Partition your list into equivalence classes such that g_i and g_j are in the same class if and only if $g_i \in \Theta(g_j)$.

$(\frac{3}{2})^n$	$(\sqrt{2})^{\lg n}$	$\lg^* n$	n^2	$(\lg n)!$
n^3	$(\lg n)^2$	$\lg(n!)$	2^{2^n}	$n^{\frac{1}{\lg n}}$
$\lg\lg n$	$n \cdot 2^n$	$n^{\lg\lg n}$	$\ln n$	2^n
$2^{\lg n}$	$(\lg n)^{\lg n}$	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$n!$	$2^{\sqrt{2\lg n}}$	n	$n \lg n$	1

Try to rank as many of the above functions as you can. You may need to refer to CLRS 3.2 or CLR 2.2 for the definition of $\lg^* n$ and help.

Sample solution. Much of the ranking is based on the following facts:
• exponential functions grow faster than polynomial functions, which grow faster than polylogarithmic functions;
• the base of a logarithm doesn't matter asymptotically, but the base of an exponential and the degree of a polynomial do matter.

The following identities are also useful:

(a) $(\lg n)^{\lg n} = n^{\lg\lg n}$ from CLRS (3.16) [3rd], (3.15) [2nd], CLR (2.9) [1st]

(b) $4^{\lg n} = n^{\lg 4} = n^2$ from CLRS (3.16) [3rd], (3.15) [2nd], CLR (2.9) [1st]

(c) $2^{\lg n} = n$

(d) $2 = n^{\frac{1}{\lg n}}$ raising (5c) above to power $\frac{1}{\lg n}$

(e) $2^{\sqrt{2\lg n}} = n^{\sqrt{\frac{2}{\lg n}}}$ raising (5d) above to power $\sqrt{2\lg n}$

(f) $(\sqrt{2})^{\lg n} = \sqrt{n}$ as $2^{(\frac{1}{2})\lg n} = 2^{(\lg n)(\frac{1}{2})} = n^{\frac{1}{2}}$

Asymptotic bounds for Stirling's formula (CLRS (3.18) [3rd], (3.17) [2nd], CLR (2.11) [1st]) are also helpful in ranking the expressions with factorials:

(g) $\lg(n!) \in \Theta(n\lg n)$ from CLRS (3.19) [3rd], (3.18) [2nd]

(h) $n! \in \Theta(n^{n+\frac{1}{2}}e^{-n})$ by dropping constants and low-order terms in (3.18) [3rd], (3.17) [2nd], (2.11)

(i) $(\lg n)! \in \Theta((\lg n)^{\lg n+\frac{1}{2}}e^{-\lg n})$ by substituting $\lg n$ for n in (5h)
$(\lg n)! \in \Theta((\lg n)^{\lg n+\frac{1}{2}}n^{-\lg e})$ from CLRS (3.16) [3rd], (3.15) [2nd], CLR (2.9) [1st]

(j) $\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}$ CLRS p58 [3rd], p55 [2nd], CLR p36 [1st]

We can now rank the functions. Those on the same line are in the same equivalence class, and those higher on the page are Ω of those below them.

2^{2^n}	
$(n+1)!$	see (5h) above
$n!$	
$n \cdot 2^n$	
2^n	
$(\frac{3}{2})^n$	
$(\lg n)^{\lg n} = n^{\lg \lg n}$	from (5a) above
$(\lg n)!$	see (5i) above
n^3	
$n^2 = 4^{\lg n}$	from (5b) above
$n \lg n$ and $\lg(n!)$	see (5g) above
$n = 2^{\lg n}$	from (5c) above
$(\sqrt{2})^{\lg n}$	see (5f) above
$2^{\sqrt{2 \lg n}}$	see (5e) above
$(\lg n)^2$	
$\ln n$	
$\sqrt{\lg n}$	
$\lg \lg n$	
$\lg^* n$	
$n^{\frac{1}{\lg n}}$ and 1	see (5d) above

6. (CLRS Exercise 3.1-2, p52 [3rd], p50 [2nd], CLR Exercise 2.1-2, p31 [1st])
Show that for any real constants a and b, where $b > 0$,

$$(n+a)^b \in \Theta(n^b).$$

Sample solution. We need to find constants $c_1, c_2, n_0 > 0$ such that:

$$\forall n \bullet n \geq n_0 \Rightarrow 0 \leq c_1 n^b \leq (n+a)^b \leq c_2 n^b.$$

Note that for $|a| \leq n$: $n + a \leq n + |a| \leq 2n$, and for $|a| \leq \frac{1}{2}n$: $n + a \geq n - |a| \geq \frac{1}{2}n$.
Thus for $n \geq 2|a|$: $0 \leq \frac{1}{2}n \leq n + a \leq 2n$.

As $b > 0$, the inequality continues to hold when all parts are raised to the power b:

$$0 \leq (\frac{1}{2}n)^b \leq (n+a)^b \leq (2n)^b \Rightarrow 0 \leq (\frac{1}{2})^b n^b \leq (n+a)^b \leq 2^b n^b.$$

Taking $c_1 = (\frac{1}{2})^b$, $c_2 = 2^b$ and $n_0 = 2|a|$ satisfies the definition for Θ.

Alternatively, as the two functions are asymptotically non-negative, we can take the limit of their ratio. If it is a non-zero constant, then $(n+a)^b \in \Theta(n^b)$.

$$\lim_{n\to\infty} \frac{(n+a)^b}{n^b} = \lim_{n\to\infty} \left(\frac{n+a}{n}\right)^b = \lim_{n\to\infty} \left(1 + \frac{a}{n}\right)^b = 1^b = 1$$

7. (Programming problem)
Algorithms for both insertion sort and merge sort have been studied. You may use any implementations that you wish.

Perform an empirical analysis of these algorithms by running them in order to determine their execution time for different inputs (e.g., the best-case and worst-case inputs for insertion sort) and different sizes of input (eg, 1000, 2000, 4000). Compare your results with the results of the theoretical analysis of these algorithms. (You may find it useful to graph your experimental results.)

You may also like to experiment with profiling the execution of the algorithms.

Sample solution. No solution provided. One useful analysis method is to insert statements into the code which count the number of operations executed, and print the counts for differing inputs. Sensible things to count when analyzing sorting algorithms are key comparisons and data moves. Consider the complexity of each of these as functions of the input size.

8. A *local minimum* of an array segment $A[lo..hi]$ is an element $A[k]$ satisfying,

$$A[k-1] \geq A[k] \leq A[k+1]$$

for $lo \leq k \leq hi$. We assume that $lo \leq hi$, and that the indices of the whole array, A, range from (at least) $lo-1$ through to $hi+1$. This allows us to also assume that

$$A[lo-1] \geq A[lo] \wedge A[hi] \leq A[hi+1]. \qquad (1)$$

This condition guarantees that a local minimum must exist; without (1) the array may be either strictly increasing or strictly decreasing, in which case there is no local minimum.

(a) Design an algorithm for finding a local minimum of the array segment $A[lo..hi]$ which is substantially faster than the obvious $O(n)$ one in the worst case. Hint: use a divide-and-conquer approach similar to binary search.

Sample solution. The pseudocode algorithm below examines the middle two elements in the array and adjusts either the low or high bound to maintain the condition that $A[lo-1] \geq A[lo]$ and $A[hi] \leq A[hi+1]$. The range of the array to be searched is reduced by one half on each iteration and the algorithm terminates when the low and high bounds are equal. In this state, $A[lo-1] \geq A[lo] \leq A[lo+1]$.

```
LocalMin($A, lo, hi$)
1  // Precondition and invariant: $lo \leq hi \wedge A[lo-1] \geq A[lo] \wedge A[hi] \leq A[hi+1]$
2  while $lo \neq hi$
3      $mid = (lo+hi)/2$      // $lo \leq mid < hi$
4      if $A[mid] \geq A[mid+1]$
5          $lo = mid+1$      // $A[lo-1] \geq A[lo]$
6      else
7          $hi = mid$         // $A[hi] < A[hi+1]$
8  // $A[lo-1] \geq A[lo] \leq A[lo+1]$
9  return $lo$
```

where $(lo+hi)$ / 2 $= \lfloor (lo+hi)/2 \rfloor$.

(b) Give an analysis of your algorithm to determine the number of comparisons of array elements required to find a local minimum of an array of size n.

Sample solution. In the sample solution given here we do a detailed analysis which illustrates the use of floors and ceilings. A useful property of floors and ceilings is $\lfloor (x+1)/2 \rfloor = \lceil x/2 \rceil$. To reason about floors and ceilings for this example, just consider the odd and even cases separately, i.e., either $x = 2k$ or $x = 2k+1$.

Note that we are only required to determine the number of comparisons of array elements. We take the size of the problem instance, n, to be the range of elements of `A` to be searched: $n = hi - lo + 1$. For $n = 1$, i.e., $lo = hi$, no comparisons of array elements are performed and hence the number of comparisons for $n = 1$, $C(1) = 0$.

If $lo < hi$ (i.e., $n > 1$) then there is a comparison made at the midpoint of the range, $mid = \lfloor (lo+hi)/2 \rfloor$, the size of the range to be searched is reduced to some new value n'. The total number of comparisons is given by, $C(n) = 1 + C(n')$. There are two cases for n':
if $A[mid] \geq A[mid+1]$ then

$$n' = hi - (mid+1) + 1 = hi - \lfloor (lo+hi)/2 \rfloor = \lfloor (hi - lo + 1)/2 \rfloor = \lfloor n/2 \rfloor;$$

if $A[mid] < A[mid+1]$ then

$$n' = mid - lo + 1 = \lfloor (lo+hi)/2 \rfloor - lo + 1 = \lfloor (hi - lo + 2)/2 \rfloor = \lceil (hi - lo + 1)/2 \rceil = \lceil n/2 \rceil.$$

Taking the larger of these two values for n' we get,

$$\begin{array}{ll} C(n) \le 1 + C(\lceil n/2 \rceil), & \text{for } n > 1 \\ C(n) \le 1 + (1 + C(\lceil \lceil n/2 \rceil /2 \rceil), & \text{for } n > 2 \\ C(n) \le 2 + C(\lceil n/2^2 \rceil) & \text{for } n > 2 \\ C(n) \le 3 + C(\lceil n/2^3 \rceil) & \text{for } n > 4 \\ \ldots & \\ C(n) \le k + C(\lceil n/2^k \rceil) & \text{for } n > 2^{k-1} \end{array}$$

Taking $k = \lceil \lg n \rceil$ we get

$$C(n) \le \lceil \lg n \rceil + C(1) = \lceil \lg n \rceil.$$

9. (See CLRS Exercise 4.4-6 p93 [3rd], 4.2-2 p72 [2nd], CLR Exercise 4.2-2, p60 [1st])
Argue that the solution to the recurrence

$$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + cn$$

where c is a constant, is $\Omega(n \lg n)$ by appealing to a recursion tree. That is, we would like a lower bound on the solution to the recurrence.

Sample solution. The recursion tree for $T(n)$ consists of a root node with cost cn, a left subtree corresponding to $T(\frac{n}{3})$ and a right subtree corresponding to $T(\frac{2n}{3})$. The left subtree consists of a node with cost $\frac{cn}{3}$, a left subtree corresponding to $T(\frac{n}{9})$, and a right subtree corresponding to $T(\frac{2n}{9})$. The right subtree of the root consists of a node with cost $\frac{2cn}{3}$, a left subtree corresponding to $T(\frac{2n}{9})$, and a right subtree corresponding to $T(\frac{4n}{9})$. This is shown in Figure 4.6 [3rd] 4.2 [2nd] of CLR(S). Try drawing the tree out yourself to another level.

If we sum the costs associated with the nodes across each level of the tree we find the sum is always cn: the root node has cost cn, so the top level costs cn; the next level has cost $\frac{cn}{3} + \frac{2cn}{3} = cn$; and the third level has cost $\frac{cn}{9} + \frac{2cn}{9} + \frac{2cn}{9} + \frac{4cn}{9} = cn$.

To get a lower bound on the cost of the tree we need a lower bound on the height of the tree for that part of it that contains complete rows of nodes. The shortest path from the root to a leaf in the recursion tree is

$$n \to (\frac{1}{3})n \to (\frac{1}{3})^2 n \to \cdots \to 1.$$

Hence the height of the tree is guaranteed to be at least $k+1$, where $(\frac{1}{3})^k n = 1$. Because $(\frac{1}{3})^k n = \frac{n}{3^k} = 1$ when $k = \log_3 n$, the height of the part of the tree in which every node has two children is $1 + \log_3 n$. Since the values at each of these levels of the tree add up to cn, the solution to the recurrence is at least $cn(1 + \log_3 n) \in \Omega(n \lg n)$.

10. (See CLRS Problem 4-1, p107 [3rd], p85 [2nd], CLR Problem 4-1, p72 [1st])
Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \le 1$. Make your bounds as tight as possible, and justify your answers. Hint: use the master theorem.

In parts (10a), (10b) and (10d) below, we are applying case 3 of the master theorem, which requires that $af(\frac{n}{b}) \le cf(n)$ for some $c < 1$. In each of these parts, $f(n)$ has the form n^d. The desired condition is satisfied because

$$af(\frac{n}{b}) = a\left(\frac{n}{b}\right)^d = (\frac{a}{b^d})n^d = (\frac{a}{b^d})f(n) \qquad (2)$$

so $af(\frac{n}{b}) \le cf(n)$ becomes $(\frac{a}{b^d})f(n) \le cf(n)$, and hence we can find a $c < 1$ to satisfy this provided $\frac{a}{b^d} < 1$.

(a) $T(n) = 2T(\frac{n}{2}) + n^3$.

Sample solution. $T(n) = 2T(\frac{n}{2}) + n^3 \in \Theta(n^3)$. This is a divide-and-conquer recurrence with $a = 2$, $b = 2$, $f(n) = n^3$, and $n^{\log_b a} = n^{\log_2 2} = n$. The quotient

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n} = n^2 \in \Omega(n^2).$$

As $2 > 0$, case 3 of the master theorem applies, and $T(n) \in \Theta(n^3)$. For case 3 we also need to show that $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$, which using (2) reduces to showing $\frac{a}{b^d} = \frac{2}{2^3} < 1$.

(b) $T(n) = T(\frac{9n}{10}) + n$.

Sample solution. $T(n) = T(\frac{9n}{10}) + n \in \Theta(n)$. This is divide-and-conquer recurrence with $a = 1$, $b = \frac{10}{9}$, $f(n) = n$, and $n^{\log_b a} = n^{\log_{\frac{10}{9}} 1} = n^0 = 1$. The quotient

$$\frac{f(n)}{n^{\log_b a}} = \frac{n}{1} = n \in \Theta(n^1).$$

As $1 > 0$, case 3 of the master theorem applies, and $T(n) \in \Theta(n)$. For case 3 we also need to show that $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$, which using (2) reduces to showing $\frac{a}{b^d} = \frac{1}{(\frac{10}{9})^1} = \frac{9}{10} < 1$.

(c) $T(n) = 16T(\frac{n}{4}) + n^2$.

Sample solution. $T(n) = 16T(\frac{n}{4}) + n^2 \in \Theta(n^2 \lg n)$. This is divide-and-conquer recurrence with $a = 16$, $b = 4$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_4 16} = n^2$. The quotient

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^2} = 1 \in \Theta(1).$$

Hence case 2 of the master theorem applies, and $T(n) \in \Theta(n^2 \lg n)$.

(d) $T(n) = 7T(\frac{n}{3}) + n^2$.

Sample solution. $T(n) = 7T(\frac{n}{3}) + n^2 \in \Theta(n^2)$. This is a divide-and-conquer recurrence with $a = 7$, $b = 3$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_3 7}$. The quotient

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^{\log_3 7}} = n^{2-\log_3 7}.$$

As $1 < \log_3 7 < 2$, we know $2 - \log_3 7 > 0$ and so $\frac{f(n)}{n^{\log_b a}} \in \Omega(n^\epsilon)$ for some $\epsilon > 0$, and hence case 3 of the master theorem applies, and $T(n) \in \Theta(n^2)$. For case 3 we also need to show that $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$, which using (2) reduces to showing $\frac{a}{b^d} = \frac{7}{3^2} < 1$.

(e) $T(n) = 7T(\frac{n}{2}) + n^2$.

Sample solution. $T(n) = 7T(\frac{n}{2}) + n^2 \in \Theta(n^{\lg 7})$. This is a divide-and-conquer recurrence with $a = 7$, $b = 2$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_2 7}$. The quotient

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^{\log_2 7}} = n^{2-\log_2 7}.$$

As $2 < \log_2 7 < 3$, we know $2 - \log_2 7 < 0$ and so $\frac{f(n)}{n^{\log_b a}} \in O(n^{-\epsilon})$ for some $\epsilon > 0$, and hence case 1 of the master theorem applies, and $T(n) \in \Theta(n^{\lg 7})$.

11. (See CLRS Problem 4-1, p107 [3rd], p85 [2nd], CLR Problem 4-1, p72 [1st])
Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers. Hint: use iteration.

(a) $T(n) = T(n - 1) + n$.

Sample solution. $T(n) = T(n-1) + n \in \Theta(n^2)$. This recurrence can be solved by iteration.

$$\begin{array}{ll}
T(n) = T(n-1) + n & \text{for } n > 1 \\
= T(n-2) + (n-1) + n & \text{for } n > 2 \\
= T(n-3) + (n-2) + (n-1) + n & \text{for } n > 3 \\
= T(n-i) + (n-(i-1)) + \cdots + (n-2) + (n-1) + n & \text{for } n > i \\
= T(n-i) + \sum_{j=0}^{i-1}(n-j) & \text{for } n > i
\end{array}$$

Taking $n - i = 1$, we have $i = n - 1$ and $T(n-i) = T(1) \in \Theta(1)$. Therefore,

$$\begin{array}{rcl}
T(n) & = & \Theta(1) + \displaystyle\sum_{j=0}^{n-2}(n-j) \\
& = & \Theta(1) + n\displaystyle\sum_{j=0}^{n-2} 1 - \sum_{j=0}^{n-2} j \\
& = & \Theta(1) + n(n-1) - \displaystyle\sum_{j=1}^{n-2} j \\
& = & \Theta(1) + n(n-1) - \dfrac{(n-2)(n-1)}{2} \\
& = & \Theta(1) + n^2 - n - \dfrac{(n^2 - 3n + 2)}{2} \\
& = & \Theta(1) + \dfrac{n^2}{2} + \dfrac{n}{2} - 1 \\
& \in & \Theta(n^2)
\end{array}$$

(b) $T(n) = T(\sqrt{n}) + 1$.

Sample solution. $T(n) = T(\sqrt{n}) + 1 \in \Theta(\lg\lg n)$. This recurrence can be solved by iteration.

$$\begin{array}{ll}
T(n) = T(n^{\frac{1}{2}}) + 1 & \text{for } n > 2 \\
= T(n^{\frac{1}{4}}) + 1 + 1 = T(n^{\frac{1}{2^2}}) + 2 & \text{for } n^{\frac{1}{2}} > 2 \\
= T(n^{\frac{1}{8}}) + 1 + 1 + 1 = T(n^{\frac{1}{2^3}}) + 3 & \text{for } n^{\frac{1}{4}} > 2 \\
= T(n^{\frac{1}{2^i}}) + i & \text{for } n^{\frac{1}{2^{i-1}}} > 2
\end{array}$$

We take the boundary condition $n = 2$ rather than $n = 1$ because no amount of repeated square-root-taking can reduce a number greater than 1 all the way down to 1. The number of times we need to iterate before reaching $n \leq 2$ is i (actually $\lceil i \rceil$), where $n^{\frac{1}{2^i}} = 2$. Solving for i,

$$\begin{array}{ll}
n^{\frac{1}{2^i}} = 2 \iff (\frac{1}{2^i})\lg n = 1 & \text{taking lg of both sides} \\
\iff 2^i = \lg n \iff i = \lg\lg n & \text{taking lg of both sides}
\end{array}$$

Hence $T(n) = \Theta(1) + \Theta(\lg\lg n) \in \Theta(\lg\lg n)$.

A more intuitive way to realise that there are $\lg\lg n$ steps is to notice that each step (taking the square root, or raising to the power $\frac{1}{2}$) halves the $\lg$ of the argument, so the number of steps to reduce $\lg$ to 1 (i.e., to $\lg 2$) is the $\lg$ of the $\lg$.

12. Solve (find an asymptotic upper bound for) the recurrence

$$T(n) = T(n-a) + T(a) + n$$

where $a \geq 1$ is a constant, by using iteration to generate a guess and then using substitution to verify it.

Sample solution. Iteration. Because a is a constant, we may assume $T(n)$ is bounded by a constant for $n \leq a$: we can take as our bound the maximum of $T(n)$ for n in the range 1 through to a. Hence $T(n) \in \Theta(1),$ for $n \leq a$. Using iteration to solve the recurrence we get

$$\begin{array}{ll}
T(n) = T(n-a) + T(a) + n & \text{for } n \geq a \\
= (T(n-2a) + T(a) + (n-a)) + T(a) + n & \text{for } n \geq 2a \\
= T(n-2a) + 2T(a) + 2n - a & \\
= (T(n-3a) + T(a) + (n-2a)) + 2T(a) + 2n - a & \text{for } n \geq 3a \\
= T(n-3a) + 3T(a) + 3n - (1+2)a & \\
= (T(n-4a) + T(a) + (n-3a)) + 3T(a) + 3n - (1+2)a & \text{for } n \geq 4a \\
= T(n-4a) + 4T(a) + 4n - (1+2+3)a & \\
= \ldots &
\end{array}$$

The ith iteration adds n, adds $T(a)$, and subtracts $(i-1) \cdot a$, so we have

$$T(n) = T(n - i \cdot a) + i \cdot T(a) + i \cdot n - a \cdot \sum_{j=1}^{i-1} j \quad \text{for } n \geq i \cdot a$$

The argument to T is less than the constant a when $i = \lfloor \frac{n}{a} \rfloor$. Ignoring the floor:

$$\begin{array}{rcl}
T(n) & = & \Theta(1) + \dfrac{n}{a} \cdot \Theta(1) + \dfrac{n}{a} \cdot n - a \displaystyle\sum_{j=1}^{\frac{n}{a}-1} j \\
& = & \Theta(n) + \dfrac{n^2}{a} - \dfrac{a}{2}(\dfrac{n}{a} - 1)(\dfrac{n}{a}) \\
& = & \Theta(n) + \dfrac{n^2}{a} - \dfrac{n^2}{2a} + \dfrac{n}{2} \\
& = & \Theta(n) + \dfrac{n^2}{a}(1 - \dfrac{1}{2}) \\
& = & \Theta(n) + \dfrac{n^2}{2a} \\
& = & \Theta(n^2)
\end{array}$$

Substitution. To show that $T(n) \in O(n^2)$, we want to show by induction that $T(n) \leq cn^2$ for some $c \geq 0$. Assume that this is true for all arguments less than some n. Later we will make sure to pick our initial conditions to cover values at least up to a.

$$\begin{array}{ll}
T(n) = T(n-a) + T(a) + n & \\
\leq c(n-a)^2 + ca^2 + n & \text{ind. hypoth.} \\
= cn^2 - 2cna + ca^2 + ca^2 + n & \\
= cn^2 - (2cna - 2ca^2 - n) & \\
\leq cn^2 & \text{provided } 2cna - 2ca^2 - n \geq 0
\end{array}$$

$$\begin{array}{ll}
2cna - 2ca^2 - n \geq 0 & \\
\Leftrightarrow c(2na - 2a^2) \geq n & \\
\Leftrightarrow c \geq \frac{n}{2na - 2a^2} & \text{provided } 2na - 2a^2 > 0 \text{, i.e., } n > a \\
\Leftrightarrow c \geq \frac{1}{2a - \frac{2a^2}{n}} &
\end{array}$$

For $n \geq 2a$, the last expression is less than or equal to $\frac{1}{a}$. Since $a \geq 1$, any $c \geq 1$ works. Pick c large enough to satisfy the initial conditions, i.e., large enough so that $T(n) < cn^2$ for $n < 2a$.