Exercise 1-1 Exercises (Page 57) 3-1, 3-2, 3-3, 3-4.

3-1.

1. Let $c = \sum_i a_i$, then $\forall n > 0$, $p(n) \leq cn^k$ if $k \geq d$. Hence $p(n) = O(n^k)$.
2. Let $c = \min_i a_i$, then $\forall n > 0$, $p(n) \geq cn^k$ if $k \leq d$. Hence $p(n) = \Omega(n^k)$.
3. Let $c_1 = \sum_i a_i, c_2 = \min_i a_i$, then $\forall n > 0$, $c_1 n^k \geq p(n) \geq c_2 n^k$ if $k = d$. Hence $p(n) = \Theta(n^k)$.
4. For any constant $c > 0$, note that $\forall n > \left(\frac{\sum_i a_i}{c}\right)^{1/(k-d)}$, $p(n) < cn^k$ if $k > d$. Hence $p(n) = o(n^k)$.
5. For any constant $c > 0$, note that $\forall n > \left(\frac{c}{a_d}\right)^{1/(d-k)}$, $p(n) > cn^k$ if $k < d$. Hence $p(n) = \omega(n^k)$.

3-2.

yes	yes	no	no	no
yes	yes	no	no	no
no	no	no	no	no
no	no	yes	yes	no
yes	no	yes	no	yes
yes	no	yes	no	yes

3-3.

$2^{2^{n+1}} = \Omega(2^{2^n})$, $2^{2^n} = \Omega((n+1)!)$, $(n+1)! = \Omega(n!)$, $n! = \Omega(e^n)$, $e^n = \Omega(n \cdot 2^n)$, $n \cdot 2^n = \Omega(2^n)$, $2^n = \Omega((3/2)^n)$, $(3/2)^n = \Omega((lgn)^{lgn})$, $(lgn)^{lgn} = \Omega(n^{lglgn})$, $n^{lglgn} = \Omega((lgn)!)$, $(lgn)! = \Omega(n^3)$, $n^3 = \Omega(n^2)$, $n^2 = \Theta(4^{lgn})$, $4^{lgn} = \Omega(nlgn)$, $nlgn = \Theta(lg(n!))$, $lg(n!) = \Omega(2^{lgn})$, $2^{lgn} = \Theta(n)$, $n = \Omega((\sqrt{2})^{lgn})$, $(\sqrt{2})^{lgn} = \Omega(2^{\sqrt{2lgn}})$, $2^{\sqrt{2lgn}} = \Omega(lg^2 n)$, $lg^2 n = \Omega(\ln n)$, $\ln n = \Omega(\sqrt{lgn})$, $\sqrt{lgn} = \Omega(\ln \ln n)$, $\ln \ln n = \Omega(2^{lg^* n})$, $2^{lg^* n} = \Omega(lg^* n)$, $lg^* N = \Omega(lg(lg^* n))$, $lg(lg^* n) = \Omega(lg^*(lgn))$, $lg^*(lgn) = \Omega(n^{1/lgn})$, $n^{1/lgn} = \Omega(1)$

An example: $f(n) = (\ln \ln n)^2$.

3-4.
a. false. example: $f(n) = n, g(n) = n^2$.
b. false. example: $f(n) = n^2, g(n) = n$.
c. true.
d. true.
e. false. example: $f(n) = 1/n$
f. true.
g. false. example: $f(n) = 2^n$.
h: true.

Exercise 1-2 Exercises (Page 85) 4-1, 4-2, 4-3, 4-4 (a,b,c,d,e,f), (page 75) 4.3-2.

4-1.
a. $\Theta(n^3)$.
b. $\Theta(n)$.
c. $\Theta(n^2 \log n)$.
d. $\Theta(n^2)$.
e. $\Theta(n^{\log 7})$.
f. $\Theta(\sqrt{n} \log n)$.

g. $\Theta(n^2)$.
h. $\Theta(\log \log n)$.

4-2.

We first count the number of 1s at the first bit. If the number of 1s (or 0s) is $n/2$, then we know the first bit of the missing integer must be 0 (or 1). We can discard all the numbers starting with 1 (or 0) and consequently the problem size is reduced by half. Repeating this procedure for all the rest bits we can detect the missing integer.

Let $T(n)$ be the total runtime. We have $T(n) = T(n/2) + \Theta(n)$ and hence $T(n) = \Theta(n)$.

4-3.

a.

1. $T(n) = T(n/2) + \Theta(1)$. $T(N) = \Theta(\log N)$.
2. $T(n) = T(n/2) + \Theta(N)$. $T(N) = \Theta(N \log N)$.
3. $T(n) = T(n/2) + \Theta(n)$. $T(N) = \Theta(N)$.

b.

1. $T(n) = 2T(n/2) + \Theta(n)$. $T(N) = \Theta(N \log N)$.
2. $T(n) = 2T(n/2) + \Theta(N)$. $T(N) = \Theta(N^2)$.
3. $T(n) = 2T(n/2) + \Theta(n)$. $T(N) = \Theta(N \log N)$.

4-4.

a. $\Theta(n^{\log(3/2)})$.
b. $\Theta(n \log \log n)$.
c. $\Theta(n^2\sqrt{n})$.
d. $\Theta(n \log n)$.
e. $\Theta(n \log \log n)$.
f. $\Theta(n)$.
g. $\Theta(n^2)$.

4.3-2.
49.

Problem 1-1 Show the following using the recursion tree method:

- (*a*). If $T(1) = c$ and $T(n) \leq T(n/2) + d$ for every $n \geq 2$ that is a power of 2, $T(n) \leq d \log_2 n + c$ whenever n is a power of 2.

  Summing over all levels we obtain $T(n) \leq d \log_2 n + c$.

- (*b*). If $T(1) = 1$ and $T(n) = 4T(n/2) + n^2$ for every $n \geq 2$ that is a power of 2, $T(n) \in \Theta(n^2 \log_2 n)$ whenever n is a power of 2. Is the above result correct if the recurrence relation is replaced by $T(1) = 1$ and $T(n) \leq 4T(n/2) + n^2$? Justify.

Note that the value in the first level is equal to which in the last level, hence $T(n) = n^2 \cdot (\log n + 1)$ and clearly it is both upper and lower bound. Yet, the above result is **not** correct if the recurrence relation is replaced by $T(1) = 1$ and $T(n) \leq 4T(n/2) + n^2$. In fact, we can only show the upperbound result, that is, $T(n) = O(n^2 \log n)$. The lowerbound result does not hold, for example, it will not violate the given condition if $T(n) = O(1)$.

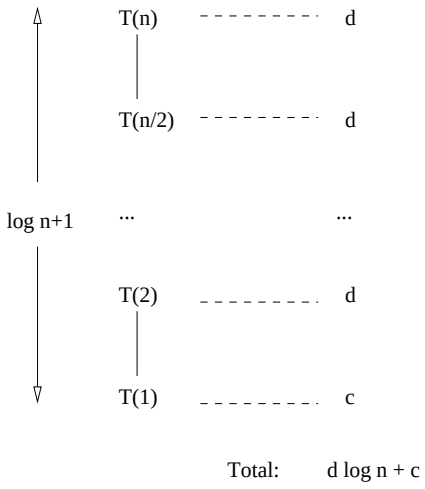

Figure 1: The recursion tree (a)

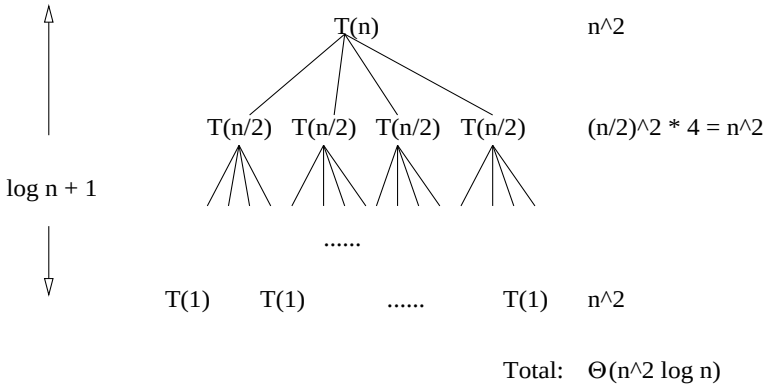

Figure 2: The recursion tree (b)

Problem 1-2 (Problem taken from the algorithm text of Kleinberg and Tardos)
You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values - so there are $2n$ values total - and you may assume that no two values are the same. You would like to determine the median of this set of $2n$ values, which we will define to be the n^{th} smallest value. However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the databases and the chosen database will return the k^{th} smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Given an algorithm that finds the median value using at most $O(\log_2 n)$ queries.

Note: Organize your answer into the following three parts. First clearly state your algorithm (in English or in pseudocode). Second, briefly argue why your algorithm is correct. And third, analyze its running time (in terms of the number of database queries that it uses). You can assume that n is a power of 2.

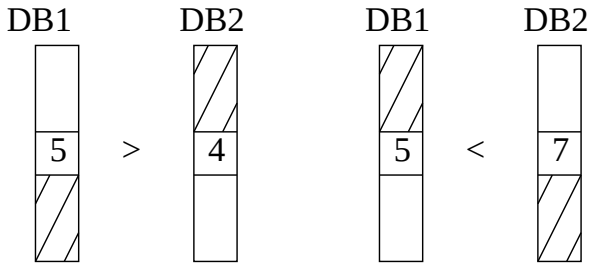

Figure 3: Left: left median > right median Right: left median < right median. In the diagram, both databases are sorted in ascending order.

Let $DB1, DB2$ be the two databases. Our algorithm is as follows:

Algorithm 1 Median finding algorithm for joint databases

$p_1 = p_2 = n/2$ (//two query pointers)
for $i = 2$ to $\log n$ **do**
  $m_1 = QueryDB(DB1, p_1)$ // get the median of DB1
  $m_2 = QueryDB(DB2, p_2)$ // get the median of DB2
  if $m_1 > m_2$ **then**
    $p_1 \leftarrow p_1 - n/2^i$ // next time, query the median of the upper half of DB1
    $p_2 \leftarrow p_2 + n/2^i$ // next time, query the median of the lower half of DB2
  else
    $p_1 \leftarrow p_1 + n/2^i$
    $p_2 \leftarrow p_2 - n/2^i$
  end if
end for
return $\min(m_1, m_2)$

In the above algorithm, p_1 and p_2 are two query pointers for both databases. We first query the medians of both databases to obtain m_1, m_2. We show the median of the joint database must be in between m_1 and m_2. To see this, observe that there are at least n records in $DB1$ and $DB2$ which are smaller than or equal to $\max(m_1, m_2)$. Hence, the median of the joint database is not greater than $\max(m_1, m_2)$. Similarly we can show the median of the joint database is not smaller than $\min(m_1, m_2)$. Then we can move the pointers p_1 and p_2 accordingly. (To visualize, in Figure 3, the shaded parts of both databases can actually be discarded.) By the end of the loop, m_1, m_2 are the n^{th} and the $(n+1)^{th}$ smallest numbers of the joint database, hence we return the smaller one among m_1, m_2.

Let $T(n)$ be the total number of queries. As each round we reduce the problem size by half using two queries, we have $T(n) = T(n/2) + 2$. Solving this recurrence we obtain $T(n) = O(\log n)$.