Chapter 3.

2. a. $M(n) = n \approx 2^b$ where $M(n)$ is the number of multiplications made by the brute-force algorithm in computing a^n and b is the number of bits in the n's binary representation. Hence, the efficiency is linear as a function of n and exponential as a function of b.

   b. Perform all the multiplications modulo m, i.e.,

$$a^i \bmod m = (a^{i-1} \bmod m \cdot a \bmod m) \bmod m \quad \text{for } i = 1, \ldots, n.$$

8.

E	X	**A**	M	P	L	E
A	X	**E**	M	P	L	E
A	E	X	M	P	L	**E**
A	E	E	M	P	**L**	X
A	E	E	L	P	**M**	X
A	E	E	L	M	**P**	X
A	E	E	L	M	P	X

9. Selection sort is not stable: In the process of exchanging elements that are not adjacent to each other, the algorithm can reverse an ordering of equal elements. The list $2'$, $2''$, 1 is such an example.

11. E, X, A, M, P, L, E

$$
\begin{array}{ccccccccccccc}
E & \overset{?}{\leftrightarrow} & X & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E \\
E & & A & & X & \overset{?}{\leftrightarrow} & M & & P & & L & & E \\
E & & A & & M & & X & \overset{?}{\leftrightarrow} & P & & L & & E \\
E & & A & & M & & P & & X & \overset{?}{\leftrightarrow} & L & & E \\
E & & A & & M & & P & & L & & X & \overset{?}{\leftrightarrow} & E \\
E & & A & & M & & P & & L & & E & & |X \\
E & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E & & \\
A & & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & P & \overset{?}{\leftrightarrow} & L & & E & & \\
A & & E & & M & & L & & P & \overset{?}{\leftrightarrow} & E & & \\
A & & E & & M & & L & & E & & |P & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & L & & E & & & & \\
A & & E & & L & & M & \overset{?}{\leftrightarrow} & E & & & & \\
A & & E & & L & & E & & |M & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & \overset{?}{\leftrightarrow} & E & & & & & & \\
A & & E & & E & & |L & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & & & & & &
\end{array}
$$

3.2

1. a. $C_{worst}(n) = n + 1$.

   b. $C_{avg}(n) = \frac{(2-p)(n+1)}{2}$. In the manner almost identical to the analysis in Section 2.1, we obtain

$$\begin{aligned} C_{avg}(n) &= [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}] + (n+1) \cdot (1-p) \\ &= \frac{p}{n}[1 + 2 + \ldots + i + \cdots + n] + (n+1)(1-p) \\ &= \frac{p}{n}\frac{n(n+1)}{2} + (n+1)(1-p) = \frac{(2-p)(n+1)}{2}. \end{aligned}$$

4. 43 comparisons.
   The algorithm will make $47 - 6 + 1 = 42$ trials: In the first one, the G of the pattern will be aligned against the first T of the text; in the last one, it will be aligned against the last space. On each but one trial, the algorithm will make one unsuccessful comparison; on one trial—when the G of the pattern is aligned against the G of the text —it will make two

comparisons. Thus, the total number of character comparisons will be $41 \cdot 1 + 1 \cdot 2 = 43$.

5. a. For the pattern 00001, the algorithm will make four successful and one unsuccessful comparison on each of its trials and then shift the pattern one position to the right:

```
0 0 0 0 0 0                                   0 0 0 0 0
0 0 0 0 1
  0 0 0 0 1
                  etc.
                                              0 0 0 0 1
```

The total number of character comparisons will be $C = 5 \cdot 996 = 4980$.

b. For the pattern 10000, the algorithm will make one unsuccessful comparison on each of its trials and then shift the pattern one position to the right:

```
0 0 0 0 0 0                                   0 0 0 0 0
1 0 0 0 0
  1 0 0 0 0
                  etc.
                                              1 0 0 0 0
```

The total number of character comparisons will be $C = 1 \cdot 996 = 996$.

c. For the pattern 01010, the algorithm will make one successful and one unsuccessful comparison on each of its trials and then shift the pattern one position to the right:

```
0 0 0 0 0 0                                   0 0 0 0 0
0 1 0 1 0
  0 1 0 1 0
                  etc.
                                              0 1 0 1 0
```

The total number of character comparisons will be $C = 2 \cdot 996 = 1,992$.

3.4

1. a. $\Theta(n!)$

For each tour (a sequence of $n+1$ cities), one needs n additions to compute the tour's length. Hence, the total number of additions $A(n)$ will be n times the total number of tours considered, i.e., $n*\frac{1}{2}(n-1)! = \frac{1}{2}n! \in \Theta(n!)$.

b. (i) $n_{max} = 16$; (ii) $n_{max} = 17$; (iii) $n_{max} = 19$; (iv) $n_{max} = 21$.

Given the answer to part a, we have to find the largest value of n such that

$$\frac{1}{2}n!10^{-10} \leq t$$

where t is the time available (in seconds). Thus, for $t = 1\text{hr} = 3.6*10^3\text{sec}$, we get the inequality

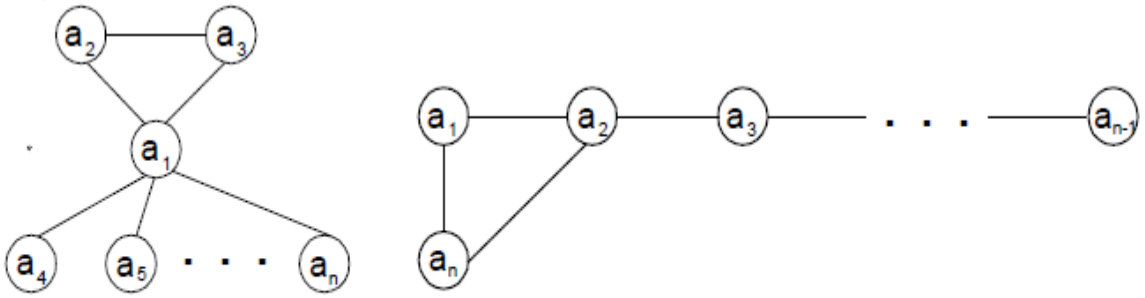
$$n! \leq 2 * 10^{10}t = 7.2 * 10^{13}.$$

The largest value of n for which this inequality holds is 16 (since $16! \approx 2.1 * 10^{13}$ and $17! \approx 3.6 * 10^{14}$).

3.5

6. a. A graph has a cycle if and only if its BFS forest has a cross edge.

b. Both traversals, DFS and BFS, can be used for checking a graph's acyclicity. For some graphs, a DFS traversal discovers a back edge in its DFS forest sooner than a BFS traversal discovers a cross edge (see example (i) below); for others the exactly opposite is the case (see example (ii) below).



(i) (ii)

8. a. Let F be a DFS forest of a graph. It is not difficult to see that F is 2-colorable if and only if there is no back edge connecting two vertices both on odd levels or both on even levels. It is this property that a DFS traversal needs to verify. Note that a DFS traversal can mark vertices as even or odd when it reaches them for the first time.

b. Similarly to part (a), a graph is 2-colorable if and only if its BFS forest has no cross edge connecting vertices on the same level. Use a BFS traversal to check whether or not such a cross edge exists.

Chapter 4.

(4) We want to ~~construct~~ construct a list, $X(n)$ that represents all the subsets of $\{a_1, a_2, a_3 \cdots a_n\}$. Read (9.3) Generating subsets on Page 146. for a detailed understanding. The algorithm using decrease-by-one is. The pseudocode

If $n = 0$ return the list, $X(n)$. // Here the list contains the empty set as its only element.

else Create recursively list, $X(n-1)$ of all the subsets of $\{a_1, \cdots, a_{n-1}\}$

Append (Add) a_n to each element of $X(n-1)$ to get List T

return $X(n)$ obtained by merging (concatenation) of $X(n-1)$ and T.

(6) Initialize a list with any of the teams.

For each of the other teams, scan the list to insert it before the first it didn't loose or at the list's end. if it lost to all the teams currently on the list.

The total # of operation in the worst case after adding is $1 + 2 + \cdots + (n-1) = (n-1)\frac{n}{2}$ (i.e each of the teams will be inserted in the end of the list).

(11) (a) The largest # of inversions, $A[i]$ $(0 \le i \le n-1)$ is $n-1-i$ this happens if $A[i]$ is greater than all the elements to the right of it. Therefore, the largest # of inversions for an entire array happens in for a strictly decreasing array. This largest # is given by the sum.

$$\sum_{i=0}^{n-1} (n-1-i) = (n-1) + (n-2) + \cdots + 1 + 0 = \frac{(n-1)n}{2}$$

The smallest # of inversions for $A[i]$ $(0 \le i \le n-1)$ is 0.

This happens if A[i] is smaller than or equal to all the elements to the right of it. Therefore, the smallest number of inversions for an entire for an entire array will be 0 for nondecreasing arrays.

(b) Assuming that all elements are distinct and that inserting A[i] in each of the i+1 possible positions among its predecessors is equally likely, we obtain the following for the expected # of key comparisons on the ith iteration

$$\frac{1}{i+1}\sum_{j=1}^{i+1} j = \frac{1}{(i+1)}\,(i+1)(i+2) = \frac{i+2}{2}$$

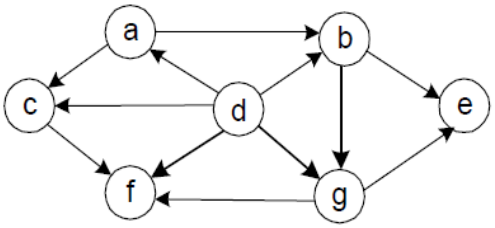
(Equally likely)

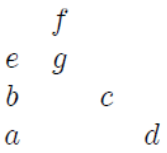
Hence for the average # of comparisons, $C_{avg}(n)$, we have

$$C_{avg}(n) = \sum_{i=1}^{n-1} \frac{i+2}{2} = \frac{1}{2}\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 = \frac{1}{2}\,\frac{(n-1)n}{2} + (n-1) \approx \frac{n^2}{4}$$

Chapter 4

1. a. The digraph and the stack of its DFS traversal that starts at vertex a are given below:





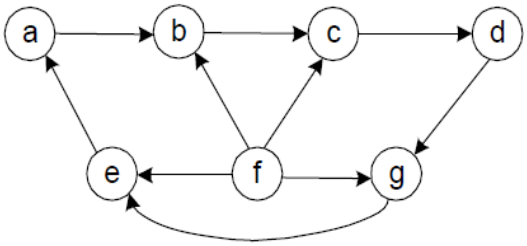
The vertices are popped off the stack in the following order:

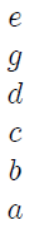
$$e\ f\ g\ b\ c\ a\ d.$$

The topological sorting order obtained by reversing the list above is

$$d\ a\ c\ b\ g\ f\ e.$$

b. The digraph below is not a dag. Its DFS traversal that starts at a encounters a back edge from e to a:

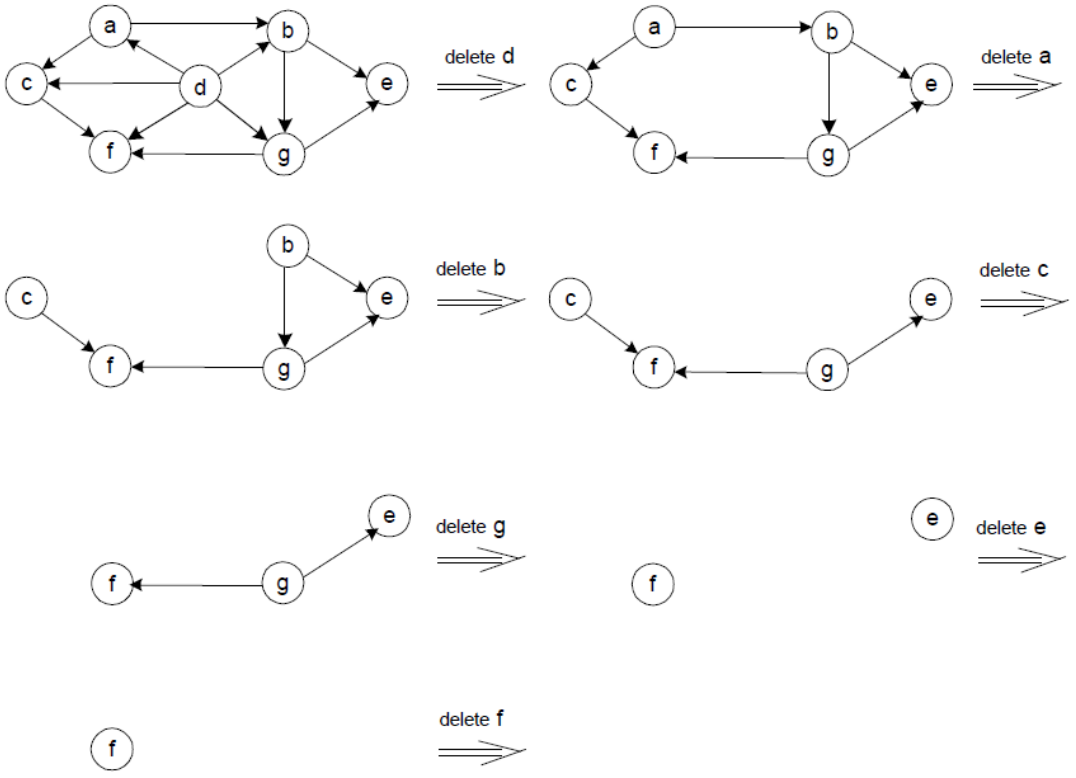




2. a. Let us prove by contradiction that if a digraph has a directed cycle, then the topological sorting problem does not have a solution. Assume that $v_{i_1}, ..., v_{i_n}$ is a solution to the topological sorting problem for a digraph with a directed cycle. Let v_{i_k} be the leftmost vertex of this cycle on the list $v_{i_1}, ..., v_{i_n}$. Since the cycle's edge entering v_{i_k} goes right to left, we have a contradiction that proves the assertion.

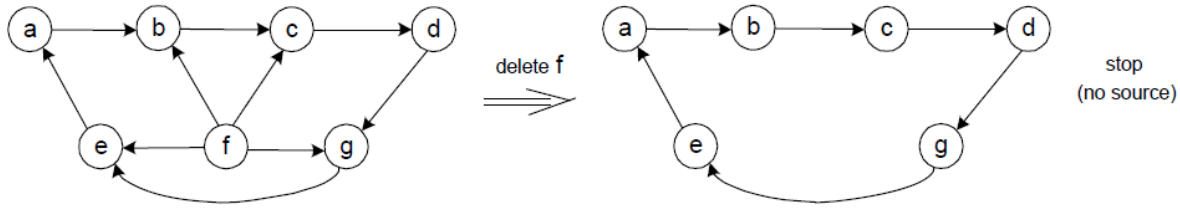
   If a digraph has no directed cycles, a solution to the topological sorting problem is fetched by either of the two algorithms discussed in the section. (The correctness of the DFS-based algorithm was explained there; the correctness of the source removal algorithm stems from the assertion of Problem 6a.)

5. a.



The topological ordering obtained is $d \quad a \quad b \quad c \quad g \quad e \quad f$.

b.



The topological sorting is impossible.

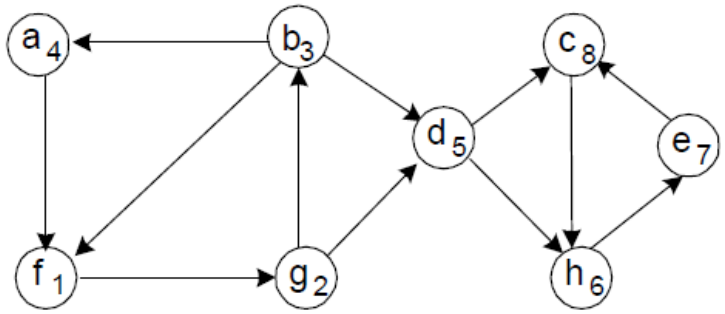
9(a)

The stack of the first DFS traversal, with a as its starting vertex, will look as follows:

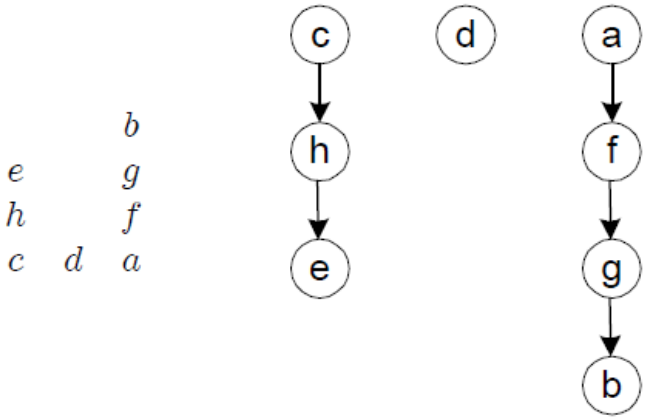
f_1
g_2 $\quad$ h_6
b_3 $\quad$ d_5 $\quad$ e_7
a_4 $\quad$ c_8

(The numbers indicate the order in which the vertices are popped off the stack.)

The digraph with the reversed edges is



The stack and the DFS trees (with only tree edges shown) of the DFS traversal of the second digraph will be as follows:



The strongly connected components of the given digraph are:

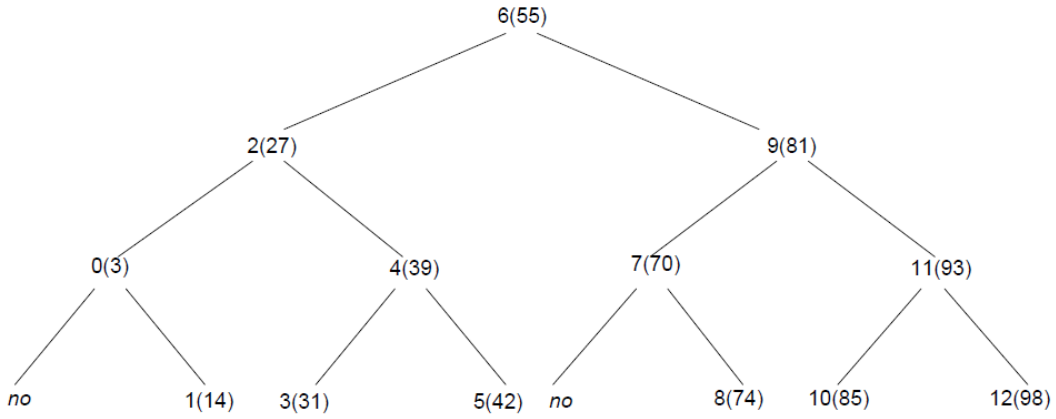
$$\{c, h, e\}, \quad \{d\}, \quad \{a, f, g, b\}.$$

b. If a graph is represented by its adjacency matrix, then the efficiency of the first DFS traversal will be in $\Theta(|V|^2)$. The efficiency of the edge-reversal step (set $B[j, i]$ to 1 in the adjacency matrix of the new digraph if $A[i, j] = 1$ in the adjacency matrix of the given digraph and to 0 otherwise) will also be in $\Theta(|V|^2)$. The time efficiency of the last DFS traversal of the new graph will be in $\Theta(|V|^2)$, too. Hence, the efficiency of the entire algorithm will be in $\Theta(|V|^2) + \Theta(|V|^2) + \Theta(|V|^2) = \Theta(|V|^2)$.

4.4

3. a. According to formula (4.5), $C_{worst}(13) = \lceil \log_2(13+1) \rceil = 4$.

b. In the comparison tree below, the first number indicates the element's index, the second one is its value:



The searches for each of the elements on the last level of the tree, i.e., the elements in positions 1(14), 3(31), 5(42), 8(74), 10(85), and 12(98) will require the largest number of key comparisons.

c. $C_{avg}^{yes} = \frac{1}{13} \cdot 1 \cdot 1 + \frac{1}{13} \cdot 2 \cdot 2 + \frac{1}{13} \cdot 3 \cdot 4 + \frac{1}{13} \cdot 4 \cdot 6 = \frac{41}{13} \approx 3.2.$

d. $C_{avg}^{no} = \frac{1}{14} \cdot 3 \cdot 2 + \frac{1}{14} \cdot 4 \cdot 12 = \frac{54}{14} \approx 3.9.$

10. a. If n is a multiple of 3 (i.e., $n \bmod 3 = 0$), we can divide the coins into three piles of $n/3$ coins each and weigh two of the piles. If $n = 3k + 1$

(i.e., $n \bmod 3 = 1$), we can divide the coins into the piles of sizes k, k, and $k+1$ or $k+1$, $k+1$, and $k-1$. (We will use the second option.) Finally, if $n = 3k+2$ (i.e., $n \bmod 3 = 2$), we will divide the coins into the piles of sizes $k+1$, $k+1$, and k. The following pseudocode assumes that there is exactly one fake coin among the coins given and that the fake coin is lighter than the other coins.

if $n = 1$ the coin is fake
else divide the coins into three piles of $\lceil n/3 \rceil$, $\lceil n/3 \rceil$, and $n - 2\lceil n/3 \rceil$ coins
    weigh the first two piles
    if they weigh the same
        discard all of them and continue with the coins of the third pile
    else continue with the lighter of the first two piles

b. The recurrence relation for the number of weighing $W(n)$ needed in the worst case is as follows:

$$W(n) = W(\lceil n/3 \rceil) + 1 \quad \text{for } n > 1, \quad W(1) = 0.$$

For $n = 3^k$, the recurrence becomes $W(3^k) = W(3^{k-1}) + 1$. Solving it by backward substitutions yields $W(3^k) = k = \log_3 n$.

c. The ratio of the numbers of weighings in the worst case can be approximated for large values of n by

$$\frac{\log_2 n}{\log_3 n} = \frac{\log_2 n}{\log_3 2 \log_2 n} = \log_2 3 \approx 1.6.$$

4.5

6. a. We can solve the inequality $\log_2 \log_2 n + 1 > 6$ as follows:

$$\begin{aligned} \log_2 \log_2 n + 1 &> 6 \\ \log_2 \log_2 n &> 5 \\ \log_2 n &> 2^5 \\ n &> 2^{32} \ (> 4 \cdot 10^9). \end{aligned}$$

b. Using the formula $\log_a n = \log_a e \ln n$, we can compute the limit as follows:

$$\begin{aligned} \lim_{n\to\infty} \frac{\log_a \log_a n}{\log_a n} &= \lim_{n\to\infty} \frac{\log_a e \ln(\log_a e \ln n)}{\log_a e \ln n} = \lim_{n\to\infty} \frac{\ln \log_a e + \ln \ln n}{\ln n} \\ &= \lim_{n\to\infty} \frac{\ln \log_a e}{\ln n} + \lim_{n\to\infty} \frac{\ln \ln n}{\ln n} = 0 + \lim_{n\to\infty} \frac{\ln \ln n}{\ln n}. \end{aligned}$$

The second limit can be computed by using L'Hôpital's rule:

$$\lim_{n\to\infty} \frac{\ln \ln n}{\ln n} = \lim_{n\to\infty} \frac{[\ln \ln n]'}{[\ln n]'} = \lim_{n\to\infty} \frac{(1/\ln n)(1/n)}{1/n} = \lim_{n\to\infty} (1/\ln n) = 0.$$

Hence, $\log \log n \in o(\log n)$.

7. a. Recursively, go to the right subtree until a node with the empty right subtree is reached; return the key of that node. We can consider this algorithm as a variable-size-decrease algorithm: after each step to the right, we obtain a smaller instance of the same problem (whether we measure a tree's size by its height or by the number of nodes).

b. The worst-case efficiency of the algorithm is linear; we should expect its average-case efficiency to be logarithmic (see the discussion in Section 4.5).