Solutions: Algorithms & Data Structures

1. (a) Sorting an array of length 1 or 2 costs 1 comparison. For arrays with more elements we have 3 recursive calls, each with an array of length $\frac{2}{3}$ of the original length. The non-recursive costs are 1 (since we have 1 comparison). This leads to the following recurrent equation for $W(n)$:

$$\begin{array}{lcll} W(n) & = & 1 & \text{for } n < 3 \\ W(n) & = & 3 \cdot W(\frac{2n}{3}) + 1 & \text{for } n \geq 3 \end{array}$$

   We apply the Master theorem: $a = 3$, $b = \frac{3}{2}$, $f(n) = 1$, and $E = \log a / \log b = \log 3 / \log \frac{3}{2}$. Since $f(n) \in O(n^{E-\epsilon})$for e.g. $\epsilon = \frac{1}{2}$ the first case of the Master theorem is applicable, so $W(n) \in \Theta(n^E)$.

   (b) Note that $E = \log 3 / \log \frac{3}{2} = {}^{1.5}\log 3 > 2$ because $1.5^2 = 2.25$. So the worst-case time complexity of *sort* is significantly worse than that of the other sorting algorithms; therefore we never prefer it.

2. Suppose x is the node containing key k. There are two possibilities:

   - x has left children. Then go 1 time down to the left, and keep going down to the right until you can go no further. Then you have found the biggest value smaller than k.
   - x has no left children. Go right upwards until you cannot go further, and then go left upward; you then reach the biggest value smaller than k. If you cannot go left upward: k was the smallest element and you yield *nill*.

   The algorithm:

```
def bstPred(x):
    if x.left != null :
        x = x.left
        while x.right != null :
            x = x.right
        return x

    else :
        y = x.parent
        while y != null and y.left == x :
            x =y
            y = y.parent
        return y
```

3. We use dynamic programming; after filling the array we have to return the most upper square with the highest number of points.

```
def maxpoints(p,n):

    R=[[0 for j in range(n+1)] for i in range(n+2)]

    for j in range(2,n+1):
        for i in range(1,n+1):
            R[i][j] = max(R[i-1][j-1]+p[i-1][j-1][R],
                          R[i+1][j-1]+p[i+1][j-1][R])

    mx=R[1][n]
```

```
for i in range(2,n+1):
    mx=max(mx,R[i][n])
return mx
```

The complexity of this algorithm is $\Theta(n^2)$.

Solutions: Discrete Mathematics

4. (a) Use the Euclidean Algorithm to compute $\gcd(1000, 444)$:

```
1000 = 2*444 + 112
 444 = 3*112 + 108
 112 = 1*108 +   4
 108 = 27* 4 +   0
```

Hence $\gcd(1000, 444) = 4$. We also know that $\gcd(1000, 444) = \min\{1000s + 444t > 0 \mid s, t \in \mathbb{Z}\}$. Because $2 < 4$, the equation $1000s + 444t = 2$ cannot have a solution in $\mathbb{Z}$.

(b) Assume that $g := \gcd(a, b)|c$, which means that $c = kg$ for some $k \in \mathbb{Z}$. As $g = \min\{sa + tb > 0 \mid s, t \in \mathbb{Z}\}$, we know that there exist $s, t \in \mathbb{Z}$ with $g = sa + tb$, hence $c = kg = (ks)a + (kt)b$, and $ks, kt \in \mathbb{Z}$.

5. (a) The characteristic polynomial of the corresponding homogeneous recurrence relation is $x^2 - 10x + 21 = (x - 3)(x - 7)$. The roots are $x_1 = 3$ and $x_2 = 7$. Hence the general solution to the homogeneous recurrence relation is

$$a_n^{(h)} = c_1 3^n + c_2 7^n \,.$$

We use as the particular solution to the inhomogeneous recurrence relation

$$a_n^{(p)} = An3^n \,,$$

(because $A3^n$ would not be linearly independent). Plugging this into the recurrence relation gives $An3^n - 10(n-1)3^{n-1} + 21(n-2)3^{n-2} = 60 \cdot 3^n$ for all n, so $An3^n(1 - \frac{10}{3} + \frac{7}{3}) + A3^n(\frac{10}{3} - \frac{14}{3}) = 60 \cdot 3^n$, hence $A = -45$. Therefore, the general solution to the inhomogeneous recurrence relation is

$$a_n = c_1 3^n + c_2 7^n - 45n3^n \,.$$

Now we have $a_0 = 2 = c_1 + c_2$, and $a_1 = -5 = 3c_1 + 7c_2 - 135$. This yields $c_1 = -29$ and $c_2 = 31$, and the solution equals

$$a_n = -29 \cdot 3^n + 31 \cdot 7^n - 45n3^n \,.$$

(b) Let a_n^k be the number of strings (with the required properties) that end on letter k, then

$$a_n = a_n^0 + a_n^1 + a_n^2 \,.$$

Now we see that $a_n^0 = a_{n-1}$, $a_n^1 = a_{n-1}^1 + a_{n-1}^2$, and $a_n^2 = a_{n-1}^1 + a_{n-1}^2$. That yields

$$a_n = a_{n-1} + (a_{n-1} - a_{n-1}^0) + (a_{n-1} - a_{n-1}^0) = 3a_{n-1} - 2a_{n-2} \,.$$

Finally, $a_1 = 3^1 = 3$, $a_2 = 3^2 - 2$(for 01 en 02) $= 7$, $a_3 = 3^3 - 3 \cdot 4$(for $01X$ and $X01$ and $02X$ and $X02$) $= 15 (= 3 \cdot 7 - 2 \cdot 3)$.

6. Let $E(s) \subseteq \delta(s)$ be the edges in $\delta(s)$ that have minimal weight (among the edges in $\delta(s)$). Since $d_e \geq 0$ for all $e \in E$, for any edge $e = \{s, v\} \in E(s)$, there can be no shorter (s, v)-path than $\{s, v\}$ itself. Hence $E(s) \subseteq D(s)$. We claim that $E(s) \cup T \neq \emptyset$ Assuming that $E(s) \cup T = \emptyset$, pick any $e = \{s, v\} \in E(s)$, and consider the (unique) (s, v)-path $P_T(s, v)$ in T. Of course $P_T(s, v)$ must contain some edge $f \in \delta(s)$, but $d_f < d_e$, because $E(s) \cup T = \emptyset$. This would be contradicting the path condition for minimum spanning trees, however. Therefore $T \cap E(s) \neq \emptyset$, and also $T \cap D(s) \neq \emptyset$.

7. (a) Take $G = K_{3,3} + \{v\}$ and connect v with any node of $K_{3,3}$. This graph is not planar, because it contains $K_{3,3}$ as a subgraph. However, $m = 10$ and $n = 7$, hence $m \leq 2n - 4$. This graph is a counterexample to the claim.

   (b) Proof: Consider any planar embedding of G. Observe that the closed walk along the boundary of any region defined by the planar embedding of G has at least 4 edges (some edges might be counted twice here). This because a bipartite graph has no cycles of odd length, hence also no cycles of length 3. (Observe here that K_2 would be a counterexample to this observation, which is the reason to require that $m > 1$). Denote by f_i the number of edges on the boundary of region i, and let r be the number of regions of the planar embedding of G. Then $2m = \sum_{i=1}^{r} f_i$, as each edge is counted exactly twice in the sum $\sum_{i=1}^{r} f_i$. Also, $\sum_{i=1}^{r} f_i \geq 4r$ as $f_i \geq 4$ for all $i = 1, \ldots, r$. Hence $r \leq \frac{1}{2}m$. Now recall the Euler formula for graph G, which says that

$$n - m + r = 2\,.$$

   Therefore, $2 = n - m + r \leq n - m + \frac{1}{2}m = n - \frac{1}{2}m$. Hence $m \leq 2m - 4$.

8. We can first compute the number of possibilities where everybody gets at least 10. This number is equal to the coefficient of x^{50} in the generating function

$$f(x) = (x^{10} + x^{11} + \cdots)^3 = x^{30}(1 + x + x^2 + \cdots)^3 = x^{30}\frac{1}{(1-x)^3}\,.$$

   This number is therefore equal to the coefficient of x^{20} in

$$\frac{1}{(1-x)^3}\,.$$

   That number equals $(-1)^{20}\binom{-3}{20} = \binom{22}{20} = 231$. Now in this count we still have those distributions where all three get 16 or more, which need to be subtracted. This can either be calculated by hand, as there are only 6 such possibilities: 16-16-18, 16-18-16, 18-16-16, 16-17-17, 17-16-17, 17-17,16. Alternatively we compute the coefficient of x^{50} in the generating function

$$f(x) = (x^{16} + x^{11} + \cdots)^3 = x^{48}(1 + x + x^2 + \cdots)^3 = x^{48}\frac{1}{(1-x)^3}\,.$$

   This number is therefore equal to the coefficient of x^2 in

$$\frac{1}{(1-x)^3}\,.$$

   That number equals $(-1)^2\binom{-3}{2} = \binom{4}{2} = 6$. The answer is therefore 231-6=225.

9. (a) false. Consider K_4, with all edges with equal weights, then there are two minimum spanning trees which are the complement of each other, hence disjoint.

   (b) false. Consider graph with three nodes $\{s, v, t\}$ and edges $(s, v), (v, t)$ with capacities $c(s, v) = 1$ and $c(v, t) = 2$. Then the only maximum flow falsifies the claim on edge (v, t).

   (c) false. Consider graph with four nodes $\{s, u, v, t\}$ and edges $(s, u), (s, v), (u, t), (v, t)$ with weights $w(s, u) = 1$ and $w(u, t) = 6$, $w(s, v) = 3$ and $w(v, t) = 4$, then there are two shortest (s, t)-paths of length 7.

   (d) true. For a proof, please refer to the tutorial sessions. Note that arguing via Kruskals's algorithm is not sufficient, even though Kruskal's algorithm computes a unique spanning tree. But potentially, there could be minimum spanning tres that can't be computed by Kruskal's algorithm... The actual proof is: Assume there exist two different MST's T_1 and T_2, then there exists at least one edge $e = \{u, v\} \in T_1 \setminus T_2$. As in T_2, u and v are also connected by a unique path $P_{T_2}(u, v)$, we know by the path condition (for T_2), that $w_f \leq w_e$ for all edges $f \in P_{T_2}(u, v)$, and since all weight are different, $w_f < w_e$ for all edges $f \in P_{T_2}(u, v)$. But now we get a contradiction to T_1 being a minimum spanning tree, because in the cut induced by $T_1 - e$, there exists at least one edge $f \in P_{T_2}(u, v)$, which is cheaper than e, contradictiong the cut condition for T_1.