1. (a) Determine the minimum number in $A$. Designate this as $Min$. This takes $O(n)$ time.
   
   (b) for $i := 1$ to $n$ do 
   $$B[i] = A[i] - Min$$
   
   (c) for $i := 1$ to $n$ do 
   $$C[i] = B[i] \cdot n^5$$
   
   (d) $C$ contains Integers in the range $[0 : n^{10}]$. These can be sorted in $O(n)$ time using Radix-Sort.

2. Here we use the Selection Algorithm done in class: $\text{Select}(a[1:n], i)$ returns the $i$-th smallest element in the array $a$.
   
   $$A := \text{Select}(X[1:n], \frac{n}{3})$$
   $$B := \text{Select}(X[1:n], \frac{2n}{3})$$
   
   Output all the elements that are in the range $[A, B]$. The total run time is $O(n)$.

3. Matrices $A$ and $B$ are partitioned into $k$ submatrices of size $n \times n$ each. In particular, $A$ is partitioned into submatrices $A_1, \ldots, A_k$ and $B$ is partitioned into $B_1, \ldots, B_k$, respectively. Then $AB =$
   
   $$\begin{pmatrix}
   A_1 & A_2 & \cdots & A_k \\
   B_1 & B_2 & \cdots & B_k
   \end{pmatrix} = \begin{pmatrix}
   A_1B_1 & A_1B_2 & \cdots & A_1B_k \\
   A_2B_1 & A_2B_2 & \cdots & A_2B_k \\
   \vdots & \vdots & \ddots & \vdots \\
   AkB_1 & AkB_2 & \cdots & AkB_k
   \end{pmatrix}$$

   $AB$ can be found by computing $A_iB_j$, for $1 \leq i, j \leq k$. Each of $A_iB_j$ can be found by using Strassen’s algorithm in $O(n^{\log_7 7})$ time. Thus the total time taken is $O(k^2n^{\log_7 7})$.

4. Let $k = \lfloor \frac{m}{w} \rfloor$. Assume w.l.o.g. that the profits of the objects are distinct. Optimal knapsack contains $k$ objects whose profits are the largest. Find the object that has the $k$th largest profit. Let its profit be $P$. This object is found using a linear time selection algorithm. Scan through the input objects and fill the knapsack with only those whose profits are $\geq P$. This algorithm takes $O(n)$ time.

   Proof of optimality: let $X = (x_1, x_2, \ldots, x_n)$ be an optimal solution and $Y = (y_1, y_2, \ldots, y_n)$ be a solution found by the algorithm above, both solutions are sorted in the descending order of $p_i$’s. Assume that $X \neq Y$, i.e., there exists an index $i$ s.t. $x_i \neq y_i$, $x_j = y_j$, $1 \leq j \leq i - 1$. Since $|X| = |Y| = k$, $i \leq k$ and due to the greedy nature of our algorithm $y_i = 1$, implying that $x_i = 0$. Also there exists a $t > k$ s.t. $x_t = 1$. Since $p_t > p_i$, replacing the $t$th object in the optimal solution by the $i$th object would improve the optimal one, a contradiction.

5. One possible minimum spanning tree has the following edges: $(3, 4), (4, 5), (2, 4), (4, 7), (1, 6)$ and $(1, 3)$. The total weight is 21.

6. At the beginning of the algorithm $\text{dist}[s] = 0; \text{dist}[1] = 4; \text{dist}[2] = \infty; \text{dist}[3] = 6; \text{dist}[4] = \text{dist}[5] = \infty$.

   In stage 1, node 1 has the minimum $\text{dist}$ value and hence is inserted into the set $S$. Nodes 2, 3 and 4 are neighbors of 1 and hence we have to check if the $\text{dist}$ values of these nodes have
to be modified. Since \( \text{dist}[2] > \text{dist}[1] + W(1, 2) \), we change \( \text{dist}[2] \) to 9. Likewise we set \( \text{dist}[3] = 5 \) and \( \text{dist}[4] = 10 \).

In stage 2, node 3 has the minimum \( \text{dist} \) value and hence is inserted into \( S \). Nodes 2 and 5 are neighbors of 3. The new \( \text{dist} \) values of these nodes become: \( \text{dist}[2] = 6; \text{dist}[5] = 8 \).

In stage 3, node 2 has the minimum \( \text{dist} \) value and it becomes a part of \( S \). Nodes 4 and 5 are neighbors of 2. The new \( \text{dist} \) value of 4 becomes 8 and the \( \text{dist} \) value of 5 does not change.

In stage 4, we have two nodes both having the same dist value. We could pick one arbitrarily and insert it into \( S \). Let 4 be this node. The \( \text{dist} \) value of 5 does not change.

In stage 5, the node 5 also enters \( S \). Algorithm terminates then.

Thus the shortest paths from \( s \) to the nodes 1, 2, 3, 4, and 5 have weights 4, 6, 5, 8, and 8, respectively.