1. Question 1

Given an undirected graph $G$ with $n$ vertices and $m$ edges. Each edge represents a highway or a flight. Let $c_e$ be the # hours needed to cross edge $e$. Suppose that
- It takes 3 extra hours to pass through the security in airport.
- No extra hour for transferring from one flight to another.

Give a polynomial time algorithm to find the fastest way to go from vertex $s$ to vertex $t$.

Algorithm.
- Let $V$ be the set of vertices in the original graph.
- Let $E_H$ and $E_F$ be the set of highway and flight edges in the original graph.
- We define a new graph $G$ as follows:
  - For each $v \in V$
    * Add two vertices $v_c$ and $v_a$ where $c, a$ denotes “city” and “airport”
    * Add an edge $(v_c, v_a)$ with cost 3
    * Add an edge $(v_a, v_c)$ with cost 0
  - For each $(u, v) \in E_H$
    * Add an edge $(u_c, v_c)$ with cost $c_{(u,v)}$
  - For each $(u, v) \in E_F$
    * Add an edge $(u_a, v_a)$ with cost $c_{(u,v)}$
- Run shortest path algorithm on $G$ from $s_c$ to $t_c$.

Runtime. $\tilde{G}$ has $O(n)$ vertices and $O(n + m)$ edges. Hence, the shortest path algorithm takes $O(m + n \log n)$.

Correctness. The length of any path in $\tilde{G}$ is exactly the time required for the trip:
- The cost of passing through security is captured by the cost of the edge $(v_c, v_a)$.
- No transfer cost is because all flight edges are on “airport” vertices.

2. Question 2

Given an undirected graph $G$ with $n$ vertices and $m$ edges. Each edge represents a highway or a flight. Let $c_e$ be the # hours needed to cross edge $e$. Suppose that
- It takes 3 extra hours to pass through the security in airport.
- No extra hour for transferring from one flight to another.
- You cannot take more than 3 flights in the whole trip.

Give a polynomial time algorithm to find the fastest way to go from vertex $s$ to vertex $t$.

Algorithm.
- Let $V$ be the set of vertices in the original graph.
- Let $E_H$ and $E_F$ be the set of highway and flight edges in the original graph.
- We define a new graph $G$ as follows:
  - For each $v \in V$
    * Add 8 vertices $\{v_{c,i}, v_{a,i}\}_{i=0}^{3}$ where $c, a$ denotes “city” and “airport” and $i$ denotes the number of flight taken
    * Add an edge $(v_{c,i}, v_{a,i})$ with cost 3 for all $i \in \{0, 1, 2, 3\}$
    * Add an edge $(v_{a,i}, v_{c,i})$ with cost 0 for all $i \in \{0, 1, 2, 3\}$
  - For each $(u, v) \in E_H$
    * Add an edge $(u_{c,i}, v_{c,i})$ with cost $c_{(u,v)}$ for all $i \in \{0, 1, 2, 3\}$
  - For each $(u, v) \in E_F$
    * Add an edge $(u_{a,i}, v_{a,i+1})$ with cost $c_{(u,v)}$ for all $i \in \{0, 1, 2\}$
    * Add the final destination $t^*_c$
    * Add an edge $(t^*_{c,i}, t^*_{c})$ with cost 0 for all $i \in \{0, 1, 2, 3\}$
- Run shortest path algorithm on $G$ from $s_{c,0}$ to $t^*_c$.

Runtime. $\tilde{G}$ has $O(n)$ vertices and $O(n + m)$ edges. Hence, the shortest path algorithm takes $O(m + n \log n)$. 


Correctness. The length of any path in $\tilde{G}$ is exactly the time required for the trip:

- The cost of passing through security is captured by the cost of the edge $(v_c, v_a)$.
- No transfer cost is because all flight edges are on "airport" vertices.

All path in $\tilde{G}$ takes at most 3 flights because each flight edge increase $i$ to $i+1$ and that $i$ starts at 0 and is capped to 3.

3. Question 3

Given a sequence of increasing integer $a_1, a_2, \cdots, a_n$. Assume there is $i$ such that $a_i = i$. Give an algorithm to find such $i$ in $O(\log n)$ time.

Algorithm.

- Define $b_i \overset{\text{def}}{=} a_i - i$ implicitly.
- Note that $b_i$ is non-decreasing.
- Run binary search on $b_i$ to find $b_i = 0$.

Runtime. Binary search takes $O(\log n)$ time ($b_i$ is only computed on fly).

Correctness. In order for binary search to work, it suffices to prove that $b_i$ is non-decreasing. This follows from

$$b_{i+1} = a_{i+1} - (i + 1) \geq a_{i} + 1 - (i + 1) = a_{i} - i = b_i.$$  

4. Question 4

Given a complete binary tree with root $r$ and $n$ vertices. Give an algorithm to find $k$ leaves of the tree in $O(k + \log n)$ time.

Algorithm.

- Let $S \leftarrow \frac{n+1}{2}$, $v \leftarrow r$
- While $S \geq 2k$
  - Pick any child $u$ of $v$
  - $v \leftarrow u$, $S \leftarrow S/2$
- Return all leaves under $v$ (using BFS/DFS on the subtree at $v$ and starts at $v$)

Correctness. First, we show that during the whole algorithm $S$ is the number of leaves under $v$.

Since the tree is complete binary tree, $n = 2^h + 2^{h-1} + \cdots + 1 = 2^{h+1} - 1$ where $h$ is the height of the tree. Hence, initially $S = \frac{n+1}{2} = 2^h$ which is exactly the number of leaves under the root. Each step, we move $v$ down by 1 step and halves $S$. This proves the claim.

When the algorithm stop walking down the tree, we have $k \leq S < 2k$. Hence, the algorithm outputs at least $k$ leaves of the tree.

Runtime. The total runtime consists of two part, the cost of walking down the tree and the cost of BFS/DFS.

For the walking down part, since the tree is complete binary tree, its height is $O(\log n)$. So is the cost.

For the BFS/DFS part, the cost is bounded by the size of the subtree at $v$, which is $O(S) = O(k)$.

Hence, the total runtime is $O(k + \log n)$.

5. Question 5

Given a weighted directed acyclic graph with $n$ vertices and $m$ edges. Give an $O(n + m)$ time algorithm to find the shortest path distance from vertex $s$ to all other vertices.

Algorithm.

- Sort vertices in topological order and rename these vertices $1, 2, 3, \cdots, n$.
- Set $d_s = 0$ and $d_u = \infty$ for all $u \neq s$.
- For $k = s, s+1, \cdots, n$
  - For every edges $(k, l)$
    - Set $d_l \leftarrow \min(d_l, d_k + \text{cost}(k, l))$.

Runtime. $O(n + m)$ time because 1) topological sort takes $O(n + m)$ time. 2) we visit every edge only once
Correctness. Induction statement: “At the beginning of step $k$, $d_k = \text{dist}(s, k)$” where dist is the shortest path distance.

**Base case:** We have $d(s) = 0 = \text{dist}(s, k)$.

**Inductive step:** Let $i_1, i_2, \cdots, i_\alpha$ be a shortest path from $s$ to $k$ (Note that $i_1 = s$ and $i_\alpha = k$). Due to the topological order, the algorithm visit the vertex $i_{\alpha-1}$ after $i_1 = s$ and before $i_\alpha = v_k$. When the algorithm do the $i_{\alpha-1}$ step, it sets

$$d_{i_\alpha} \leftarrow \min(d_{i_\alpha}, d_{i_{\alpha-1}} + \text{cost}(i_{\alpha-1}, i_\alpha)).$$

Hence, we have

$$d_k = d_{i_\alpha} \leq d_{i_{\alpha-1}} + \text{cost}(i_{\alpha-1}, i_\alpha)$$

$$= \text{dist}(s, i_{\alpha-1}) + \text{cost}(i_{\alpha-1}, i_\alpha) \quad \text{(induction hypothesis)}$$

$$= \text{dist}(s, i_\alpha) \quad (i_1, i_2, \cdots, i_\alpha \text{ is a shortest path from } v_s \text{ to } v_{i_\alpha})$$

$$= \text{dist}(s, k)$$

Also, $d_k \geq \text{dist}(s, k)$ since the algorithm finds a path from $s$ to $k$ with distance $d_k$.

6. **Question 6**

Given a connected graph with $n$ vertices and $m$ edges with $m \geq n$. Give an $O(n)$ time algorithm to find a cycle.

**Algorithm.**

- Pick arbitrary $n$ edges from the graph and call the new graph $\tilde{G}$.
- Use BFS/DFS to find a forest $F$ on $\tilde{G}$.
- Go over all edges in $\tilde{G}$ to find $e \notin F$.
- Output the cycle on $F + e$

**Runtime.** $\tilde{G}$ has $n$ edges. So, BFS/DFS takes $O(m + n) = O(n)$ time.

**Correctness.** Since $F$ has $\leq n - 1$ edges and $\tilde{G}$ has $n$ edges, there must be $e \in \tilde{G}$ that is not in $F$. Hence, we can find such $e$. Let $a$ and $b$ be the end point of $e$ and let $p$ be the first common ancestor of $a, b$. Then, the path $a \rightarrow p \rightarrow b \rightarrow a$ is a cycle.

**Remark.** One can solve this problem using DFS directly also. You simply stop DFS whenever you find a cycle. You can prove that this algorithm takes $O(n)$ time.