1. (a) **Algorithm:**

   AddMST(G, w, T, e_1, w_1):
   Use BFS to find the path P from u to v in T (where $\{u, v\} = e_1$).
   Let e_0 be an edge on P with maximum weight.
   if $w(e_0) > w_1$:
   return $T - \{e_0\} \cup \{e_1\}$
   else:
   return T

   Runtime: BFS takes time $\Theta(n+m)$ (where $n = |V|$ and $m = |E|$, as usual); finding e_0 takes time $\mathcal{O}(m)$; total time is $\Theta(n+m)$.

   Correctness: Because edge e_1 is the only difference between G and G_1, it is the only edge whose addition may result in a different MST from T. This happens only when e_1 can be swapped with some edge in T with higher weight: exactly what the algorithm does.

   (b) **Algorithm:**

   DelMST(G, w, T, e_0):
   if $e_0 \notin T$:
   return T
   else:
   Let $e_0 = \{u, v\}$.
   Run BFS on the edges of $T - \{e_0\}$, starting from u;
   assign colour *white* to every vertex encountered.
   Run BFS on the edges of $T - \{e_0\}$, starting from v;
   assign colour *black* to every vertex encountered.
   Loop over every edge in $E - \{e_0\}$ to find a minimum-weight edge e_1
   with one *white* endpoint and one *black* endpoint.
   if there is no such edge e_1:
   return NIL
   else:
   return $T - \{e_0\} \cup \{e_1\}$

   Runtime: BFS takes time $\Theta(n+m)$ (where $n = |V|$ and $m = |E|$, as usual); finding e_1 takes time $\mathcal{O}(m)$; total time is $\Theta(n+m)$.

   Correctness: Because edge e_0 is the only difference between G and G_1, it is the only edge whose removal may result in a different MST from T. From the proof of correctness of Kruskal's algorithm, we know that it is always "safe" to add an edge of minimum weight between two connected components (while constructing a MST): exactly what the algorithm does.

2. **Step 0:** *Recursive Structure.*

   Suppose $j_1, j_2, \ldots, j_\ell$ is an optimum drilling path. Then $j_2, j_3, \ldots, j_\ell$ is an optimum drilling path starting at one of the coordinates $(2, j_1 - 1), (2, j_1), (2, j_1 + 1)$ and with maximum drill hardness $d - H[1, j_1]$—if there were a better path starting from one of those coordinates and with the same maximum hardness, we could follow it after block $(1, j_1)$ to get more gold overall.

   Step 1: *Array Definition.*

   Let $M[i, j, h]$ denote the maximum amount of gold that can be drilled starting from coordinates (i, j) with drill hardness h, for $1 \leqslant i \leqslant m+1$, $0 \leqslant j \leqslant n+1$ and $0 \leqslant h \leqslant d$.

Step 2: *Recurrence Relation.*

For $0 \leqslant h \leqslant d$:

- $M[i,0,h] = M[i,n+1,h] = -\infty$ for $1 \leqslant i \leqslant m+1$ (regions outside of the geological survey cannot be drilled—setting $M = -\infty$ ensures that the drilling path does not stray outside of the surveyed region);
- $M[m+1,j,h] = 0$ for $1 \leqslant j \leqslant n$ (no gold is accessible below depth m);
- for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$,
  - $M[i,j,h] = 0$ if $h < H[i,j]$ (not enough hardness to drill block (i,j)),
  - $M[i,j,h] = G[i,j] + \max\left\{ M\big[i+1,j-1,h-H[i,j]\big], M\big[i+1,j,h-H[i,j]\big], M\big[i+1,j+1,h-H[i,j]\big] \right\}$ if $h \geqslant H[i,j]$, (get the gold from block (i,j) and do the best possible with the remaining drill hardness, starting one block below (i,j)).

Step 3: *Iterative Algorithm.*

```
for h ← 0,1,...,d:
    # Fill in values for depth m+1.
    M[m+1,0,h] ← −∞
    M[m+1,n+1,h] ← −∞
    for j ← 1,2,...,n:
        M[m+1,j,h] ← 0
    # Compute values from deepest to shallowest level.
    for i ← m,m−1,...,1:
        M[i,0,h] ← −∞
        M[i,n+1,h] ← −∞
        for j ← 1,2,...,n:
            if h < H[i,j]:
                M[i,j,h] ← 0
            else:
                M[i,j,h] ← G[i,j] + max{ M[i+1,j−1,h−H[i,j]],
                                         M[i+1,j,h−H[i,j]],
                                         M[i+1,j+1,h−H[i,j]] }
```

Runtime: $\Theta(dmn)$. This is pseudopolynomial time because of d.

Step 4: *Solution Reconstruction.*

```
h ← d    # current hardness
ℓ ← 1    # current depth
Find j_ℓ ∈ {1,...,n} that maximizes M[ℓ,j_ℓ,h].    # start of drilling path
while M[ℓ,j_ℓ,h] > 0:
    # It's possible to get more gold starting from current coordinates (ℓ,j_ℓ) with drill
    # hardness h: keep block (ℓ,j_ℓ) on the drilling path and figure out the next block.
    h ← h − H[ℓ,j_ℓ]
    ℓ ← ℓ + 1
    Find j_ℓ ∈ {j_{ℓ−1} − 1, j_{ℓ−1}, j_{ℓ−1} + 1} that maximizes M[ℓ,j_ℓ,h].
return j_1, j_2, ..., j_{ℓ−1}
```

Additional runtime: $\Theta(n+m)$ ($\Theta(n)$ to find $j_1 \in \{1,\ldots,n\}$ and $\Theta(m)$ to find each successive j_ℓ).

3. **Algorithm:**
   1. From the input, extract the following information:
      - List of CPOs: $[c_1, c_2, \ldots, c_m]$.
      - Maximum number of exams for each CPO: $[e_1, e_2, \ldots, e_m]$.
      - List of exam periods: $[p_1, p_2, \ldots, p_n]$.
      - Size of each exam pool (*already* rounded up): $[\ell_1, \ell_2, \ldots, \ell_n]$.
      - List of exam *days*: $[d_1, d_2, \ldots, d_h]$.
   2. Create network $N = (V, E)$ where V contains the following vertices:
      - source s and sink t;
      - one vertex for each CPO: $\{c_1, c_2, \ldots, c_m\}$;
      - one vertex for each exam period: $\{p_1, p_2, \ldots, p_n\}$;
      - a vertex $a_{i,k}$ for each CPO c_i and exam day d_k;

      and E contains the following edges:
      - (s, c_i) for each c_i, with capacity $c(s, c_i) = e_i$;
      - $(c_i, a_{i,k})$ for each $a_{i,k}$, with capacity $c(c_i, a_{i,k}) = 2$;
      - $(a_{i,k}, p_j)$ for each $a_{i,k}$ and p_j such that CPO c_i is available during exam period p_j and exam period p_j is on day d_k, with capacity $c(a_{i,k}, p_j) = 1$;
      - (p_j, t) for each p_j, with capacity $c(p_j, t) = \ell_j$.
   3. Find a maximum flow in network N (using the Edmonds-Karp algorithm, for example).
   4. Assign CPO c_i to exam period p_j iff $f(a_{i,k}, p_j) = 1$, where p_j is on day d_k.

Runtime: Note that $n/3 \leqslant h \leqslant n$ (there are at most three exam periods on each exam day). Creating the network takes time $\Theta(m)$ (for vertices c_i and edges (s, c_i)) + $\Theta(n)$ (for vertices p_j and edges (p_j, t)) + $\Theta(mh) = \Theta(mn)$ (for vertices $a_{i,k}$ and all related edges). This yields a network with $\Theta(mn)$ vertices and $\Theta(mn)$ edges.

Running the Edmonds-Karp algorithm takes time $\Theta((mn)(mn)^2) = \Theta((mn)^3)$.

Generating the assignment of CPOs to exam periods takes time $\Theta(mn)$ (each edge $(a_{i,k}, p_j)$ is examined once).

The total time is $\Theta((mn)^3)$.

Correctness: Consider an assignment of CPOs to exam periods that meets all the problem requirements. Then there is a corresponding flow f in N with $|f|$ = sum of the sizes of every exam pool, as follows:

- $f(s, c_i)$ = number of exam periods assigned to CPO c_i (guaranteed $\leqslant e_i$);
- $f(c_i, a_{i,k})$ = number of exam periods assigned to CPO c_i on day d_k (guaranteed $\leqslant 2$);
- $f(a_{i,k}, p_j) = 1$ if CPO c_i is assigned to exam period p_j (0 otherwise);
- $f(p_j, t)$ = size of the pool for exam period p_j (guaranteed $\leqslant \ell_j$).

Moreover,

- $f^{\text{in}}(c_i) = f^{\text{out}}(c_i)$ = total number of exam periods assigned to CPO c_i,
- $f^{\text{in}}(a_{i,k}) = f^{\text{out}}(a_{i,k})$ = number of exam periods assigned to CPO c_i on day d_k,
- $f^{\text{in}}(p_j) = f^{\text{out}}(p_j)$ = total number of CPOs assigned to exam period p_j.

This implies that the maximum flow in N is at least as large as the sum of the sizes of every exam pool.

Conversely, suppose that f is a valid *integer* flow in N. Then there is an assignment of CPOs to exam periods where the sum of the sizes of every exam pool = $|f|$, as follows: assign CPO c_i to exam period p_j iff $f(a_{i,k}, p_j) = 1$ (where p_j is on day d_k). This assignment satisfies each of the problem constraints:

- no CPO c_i is assigned to more than e_i exam periods because $c(s, c_i) = e_i$;

- no CPO c_i is assigned to more than two exam periods on the same day because $c(c_i, a_{i,k}) = 2$;
- no exam period p_j is assigned more than ℓ_j CPOs because $c(p_j, t) = \ell_j$.

This implies that the maximum sum of the sizes of every exam pool is at least as large as the maximum flow in N.

Hence, maximum flow in N = maximum sum of the sizes of every exam pool. Since the maximum flow cannot be larger than $\ell_1 + \cdots + \ell_n$ (total capacity into t), finding a maximum flow will yield an assignment of CPOs to exam periods that fills every exam pool as much as possible.