1 Short Questions $[2 + 2 + 2 = 6$ marks$]$

Solution sketch:

(i) We have $\log_2 f(n) = (\log^2 n)$ and $\log_2 g(n) = 1.5\sqrt{\log_2(n)}\log_2 n$.

For n large enough, $\log_2 g(n) < \log_2 f(n)$. Further, $\lim_{n\to\infty} \frac{\log_2 g(n)}{\log_2 f(n)} = 0$. Thus, $g(n) = o(f(n))$.

This implies, $g(n) = O(f(n))$ and $f(n) \neq O(g(n))$.

(ii) Let $B = A^3$, where A is adjacency matrix of size $n \times n$.

For any $i \in [1, n]$, we have $B[i, i] = \sum_{j,k=1}^{n} A[i, j]A[j, k]A[k, i]$.

So, $B[i, i] = 2\times$ the number of triangles having i as one endpoint.

Therefore, the number of triangles in G is $\frac{\sum_{i=1}^{n} B[i,i]}{6}$. Using fast matrix multiplication, we can compute B in $O(n^{2.38})$ time, thus the number of triangles can be computed in $O(n^{2.38})$ time.

Alternate solution:
Let $B = A^2$, where A is adjacency matrix of size $n \times n$.

For any distinct $i, j \in [1, n]$, we have $B[i, j] = \sum_{k=1}^{n} A[i, k]A[k, j]$ =number of paths of length 2 from i to j. Using fast matrix multiplication, we can compute B in $O(n^{2.38})$ time.

Now, let $s = \sum_{(i,j)\in E} B[i, j]$.

Then, $s = 3\times$ the number of triangles in G.

(iii) The problem is not NP-complete. This is because k is 10 which is a constant. So, we can check for all ${}^{n}C_{10}$ subsets if they form a vertex cover. This will take $O(n^{10}m) = O(n^{12})$ time.

Alternatively, an algorithm of $O(2^{10}(m + n))$ time complexity also exists. See below.

```
if G has no edges then Return true;
if k = 0 then return false;
(u, w) ← an arbitrary edge of G;
if Vertex-Cover(G − u, k − 1) then Return true;
if Vertex-Cover(G − w, k − 1) then Return true;
Return false;
```

Algorithm 1: *Vertex-Cover(G, k)*

2 Dynamic Programming [5 marks]

Solution sketch:

We assume that $M[i, j] = 1$ iff there is a coin at cell (i, j). Compute a matrix A in $O(n^2)$ time as follows.

Let A be an $n \times n$ matrix whose entries are intialized to 0.;
$A[1, 1] = M[1, 1]$;
for $i = 2$ *to* n **do**
    $A[i, 1] = A[i - 1, 1] + M[i, 1]$;
end
for $j = 2$ *to* n **do**
    $A[1, j] = A[1, j - 1] + M[1, j]$;
end
for $i = 2$ *to* n **do**
    for $j = 2$ *to* n **do**
        $A[i, j] = \max(A[i, j - 1], A[i - 1, j]) + M[i, j]$;
    end
end
Return $A[n, n]$;

Claim: For $i, j \geqslant 1$, $A[i, j]$ stores the maximum possible number coins that can be collected when reaching cell (i, j).

We can compute the optimal route to any cell (i, j) using A in $O(n)$ time as follows.

if $i = 1$ **then**
    Return $(1, 1) \circ (1, 2) \circ \cdots \circ (1, j - 1) \circ (1, j)$;
else if $j = 1$ **then**
    Return $(1, 1) \circ (2, 1) \circ \cdots \circ (i - 1, 1) \circ (i, 1)$;
end
if $A[i, j] = A[i, j - 1] + M[i, j]$ **then**
    Return COMPUTE-ROUTE$(i, j - 1) \circ (i, j)$;
else
    Return COMPUTE-ROUTE$(i - 1, j) \circ (i, j)$;
end

Algorithm 2: COMPUTE-ROUTE(i, j)

3 Max Flows [$3 + 3 = 6$ marks]

Solution sketch:

Let f be an (s,t)-max-flow of G. Let S be vertices reachable from s in G_f. Let T be vertices having path to t in G_f.

Claim 1: (S, S^c) and (T^c, T) are (s,t)-min-cuts.

Proof: We will prove claim for (S, S^c). Note that all edges from S to S^c are fully saturated, and edge in reverse direction are carrying zero flow. So, $c(S, S^c)$ is same as flow passing from S to S^c (which is same as (s,t)-flow value). Thus, (S, S^c) must be a min-cut as its capacity is same as (s,t)-max-flow value.

Claim 2: If $(x,y) \notin S \times T$ then (s,t)-max-flow in unchanged.

Proof: Either (S, S^c) or (T^c, T) is still an (s,t)-cut.

Claim 3: For $(x,y) \in (S \times T) \setminus E$, on addition of edge $(x,y) \in (S \times T) \setminus E$, there is path from s to t in G_f, which implies the (s,t)-max-flow increases by 1.

Proof: (i) there is path from s to x in G_f, (ii) there is path from y to t in G_f.

Remark: As the s to t path is computable in $O(m+n)$ time in G_f, the time to compute updated flow in $O(m+n)$.

Part (a) Compute (s,t)-max-flow f, and sets S, T described above. This takes $O(mn)$ time. Return $(S \times T) \setminus E$.

Part (b) Compute sets S, T. Recall that if $(x,y) \notin S \times T$ then (s,t)-max-flow in unchanged. If $(x,y) \in (S \times T) \setminus E$, then the new flow is computable in $O(m+n)$ time as described above.

4 NP completeness [7 marks]

Solution sketch:
(1) Let $H = (V, E)$ be an instance of vertex-cover with n vertices and m edges.

(2) Compute a graph G with $m + n$ vertices such that:
Layer 1 has n vertices ($\{x_v \mid v \in V\}$), and
Layer 2 has m vertices ($\{x_e \mid e \in E\}$).

(3) The edge set of G is as follows:
- Connect each pair of vertices in layer 1 by an edge.
- For each x_e in layer 2 connect x_e with x_u and x_v, where u and v are endpoints of e.

(4) Define "S" as the set of all vertices in layer 2.

(5) Set $k = m + \alpha$.

(6) **Claim:** G has a vertex cover of size α iff H has a tree covering S with $m + \alpha$ vertices.

Proof: Let us suppose G has a vertex-cover $W = \{w_1, \ldots, w_\alpha\}$ of size α. Then H has a tree covering set S of size $m+\alpha$: take path $(w_1, \ldots, w_\alpha)$ in layer 1, and connect each vertex in layer 2 to some vertex in set W.

Now, let us suppose there is a tree in H containing set S with $m + \alpha$ nodes. Such a tree contains all vertices of layer 2, and some α vertices of layer 1 (say U). Then U must be a vertex cover in G.

Alternate Solution:
(2) Compute a graph G with $m + n + 1$ vertices such that:
Layer 0 has a single vertex (say s),
Layer 1 has n vertices ($\{x_v \mid v \in V\}$), and
Layer 2 has m vertices ($\{x_e \mid e \in E\}$).

(3) The edge set of G is as follows:
- Connect each vertex in layer 1 to s.
- For each x_e in layer 2 connect x_e with x_u and x_v, where u and v are endpoints of e.

(5) Set $k = m + \alpha + 1$.

(6) **Claim:** G has a vertex cover of size α iff H has a tree covering S of size $m + \alpha + 1$.
Proof: Similar to above

5 Divide and Conquer

Solution sketch:
For each p, q ($p \leqslant q$) compute an array A such that

$$A[k] = M[p,k] + M[p+1,k] + \cdots + M[q-1,k] + M[q,k].$$

Use divide an conquer approach to find subarray of A of largest sum as follows:

1. Recursively search in $A[1, n/2]$,
2. Recursively search in $A[1 + n/2, n]$,
3. Find a sub array containing $A[n/2]$ and $A[1 + n/2]$ of largest sum as follows:

Let L, R be two arrays of size n whose entries are intialized to 0;
for $i = (n/2)$ *to* (1) **do** $L[i] = L[i+1] + A[i]$;
for $j = (1 + n/2)$ *to* (n) **do** $R[j] = R[j-1] + A[j]$;
$i_0 = \arg\max_{1 \leqslant i \leqslant n/2} L[i]$;
$j_0 = \arg\max_{1+n/2 \leqslant j \leqslant n} R[j]$;
Return $(A[i_0] + \cdots + A[n/2] + A[1 + n/2] + \cdots + A[j_0])$;

Argument:

For $i \leqslant n/2$, $L[i]$ stores the sum $A[i] + A[i+1] + \cdots + A[n/2]$.

For $j > n/2$, $R[j]$ stores the sum $A[1 + n/2] + A[2 + n/2] + \cdots + A[j]$.

So, the sub array containing $A[n/2]$ and $A[1 + n/2]$ of largest sum has sum as

$$\max_{1 \leqslant i \leqslant n/2} L[i] + \max_{1+n/2 \leqslant j \leqslant n} R[j].$$

The recurrence relation is $T(n) = 2T(n/2) + O(n)$. So, the time complexity for any $p, q \in [1, n]$ is $O(n \log n)$. The total time complexity of algorithm is therefore $O(n^3 \log n)$.

OR

Claim 1: For each $s \in S$, the shortest cycle through s in G is computable in $O(n)$ time.

Proof: For each node $i (\neq s)$, define $\textsc{label}(i)$ as the child of s lying on $\text{BFS}\textsc{path}(s,i)$. The labels are computable in $O(m)$ time. Let A be the set of all non-tree edges (i,j) satisfying that i,j have no common ancestor other than s. Such a set is computable in $O(m)$ time as an edge (i,j) can lie in A iff $\textsc{label}(i) \neq \textsc{label}(j)$. Let $(i_0, j_0) \in A$ be an edge for which $\textsc{depth}(i_0) + \textsc{depth}(j_0)$ is smallest, then $\text{BFS}\textsc{path}(s, i_0) :: (i_0, j_0) :: \text{BFS}\textsc{path}(j_0, s)$ is a shortest cycle. **(This must be proved)**.

Algorithm:
1. Partition the input graph G into three subsets S, A, B according to the planar separator theorem.
2. Recursively search for the shortest cycles in induced graphs $G[A]$ and $G[B]$.
3. Use BFS algorithm to find, for each vertex $s \in S$, the shortest cycle through s in G.
4. Return the shortest of the cycles found by the above steps.

Time complexity: We have $T(n) = T(n_1) + T(n_2) + O(n\sqrt{n})$, where n_1, n_2 sum upto n and are both bounded by $2n/3$. So time complexity is $O(n^{1.5} \log n)$ as depth of recursion is $O(\log n)$.