Q1. Disconnecting Edges

(a) Take any edge, \( e = (u, v) \), of \( G \) not in \( T \); \( u \) and \( v \) are connected in \( T \): consider the lowest common ancestor \( a \) of \( u \) and \( v \) (\( a \) can be \( u \) or \( v \) itself), there is a unique path from \( u \) to \( v \) by concatenating the path \( p_u \) from \( u \) to \( a \) and \( p_v \) from \( a \) to \( v \). Adding \( e \) into \( T \), an incremental cycle is introduced by concatenating \( p_u, p_v \) and \( e \). Because \( a \) is unique, such cycle is unique.

Along an incremental cycle all vertices are connected, and removing any one edge in this cycle will not disconnect these vertices. Therefore, all edges pertaining to an incremental cycle are not disconnecting edges.

Consider all other edges not in any incremental cycles, because of the definition of tree they are disconnecting edges in \( T \), and between each pair of vertices there is one unique simple path in \( T \). Take any one such edge, \( e' = (r, s) \), then after its removal \( r \) and \( s \) must be disconnected also in \( G \) because \( e' \) does not belong to any cycles.

Therefore all edges not in an incremental cycle must be disconnecting edges.

(b) First, traverse the whole tree and mark all vertices in left subtree of root as “left” and all in right as “right”. This takes linear time. Then go over all added edges and check if one end is “left” while another “right”. Take out all such edges in linear time.

For each of such edges, \( e = (u, v) \), of \( G \) not in \( T \), repeat the following:

Follow parent pointers of \( u \) and \( v \) until either (1) their lowest common ancestor or (2) some visited parent is reached, whichever comes earliest.

Along the path mark all vertices as visited and put all edges into result set.

All iterations together considered each edge and each vertex at most once, so the runtime is linear, too.

(c) We give a divide-and-conquer algorithm to find all disconnecting edges.

We consider the tree as the root, its left subtree and right subtree. The subproblem is to recursively find the sets, \( S_l \) and \( S_r \), of disconnecting edges for each subtree. (The recursion ends when there are less then four vertices in the subtree.) Then, if there are no edges connecting vertices on opposite sides of the root, the result is \( S_l \cup S_r \); else, use the algorithm in (b) and find all non-disconnecting edges \( S_n \), and the result is \( (S_l \cup S_r) \setminus S_n \).

As for the runtime \( T(.) \), the recursive relation is:
\[
T(n) = 2T(n/2) + O(n), \quad n = |V| + |E|
\]

According to the Master Theorem, \( T(n) = O(n\log(n)) \), i.e., \( O(|V|+|E|\log(|V|)) \).
Q2. Counterfeit Detection

We separate $n$ coins evenly into $2^k$ groups. If $n \leq 2^k$, we check each coin against a reference coin for genuineness, which takes $O(2^k)$ time. For the case when $n$ is not dividable by $2^k$, each group contains $\lceil n/2^k \rceil$ coins and the remainder coins can be weighted separated against reference coins to check for counterfeit ones; such corner case takes $O(k)$ time. Then for the $2^k$ groups, weight each group against $\lceil n/2^k \rceil$ reference genuine coins; if the weight is equal disregard all coins in this group, because all of the coins must be genuine; else the weight can only be lighter, then this group must contain at least one counterfeit coins so we preserve the whole group for future processing. Weighting $2^k$ groups takes $O(k)$ time. Notice that there are $k$ coins in $2^k$ groups of coins, at least $k$ groups of coins are all genuine, hence after one round of weightings at least half of the target coins are disregarded. We then take the remaining coins, put evenly into $2^k$ groups and repeat the above process, until there is 1 coin left for each group, in which case we only need to test that coin against one reference coin. There are at most $\lceil \log(n) \rceil + 1$ rounds of grouping and weighting. Given each round takes $O(k)$ time, the total runtime is $O(k\log(n))$.

Q3. Unbalanced Recurrence Recursions

(a) Consider the function $f(x) = ax^d + bx^e$ ($a, b \in (0, 1)$). $f(x)$ is a continuous monotonically decreasing function. When $x = 0, f(x) = 2$; when $x \to +\infty, f(x) \to 0$. Therefore there must exist one value of $x = r \in (0, 2)$ such that $f(r) = 1$.

For (b) and (c), without loss of generality, let us assume that $a \leq b$. This way we have $\min(a, b) = a$ and $\max(a, b) = b$.

(b) Proof by induction:
We want to prove that if $r < d$, then $\exists C > 0 \ \forall x \geq 0 \ T(x) \leq Cx^d$

Base case:
For $1 < x \leq 1/b$, $ax \leq bx \leq 1$ and $T(x) \leq T(ax) + T(bx) + x^d \leq 1 + 1 + 1/b^d \leq 2 + 1/b^d$.

Let us set $\partial = 2 + 1/b^d, \beta = 1/(1 - a^d - b^e)$. Because $d > r$, $a^d + b^e < a^r + b^r = 1$,
and thus $\beta > 0$. Also, $\partial > 2$. So both $\partial$ and $\beta$ are positive numbers. (We will use $\beta$ later in inductive steps.)

Let us choose $C = \max(\partial, \beta)$. Then $C \geq \partial$. In the given range of $x$, $x^d \geq 1$. So:

$\exists C > 0 \ \forall x \in (1, 1/b], T(x) \leq 2 + 1/b^d = \partial \cdot 1 \leq C \cdot x^d$

Inductive steps:
Assume for some \( y > 1/b \), \( \forall x \in (1, y], T(x) \leq Cx^d \),

Let us consider \( \forall x \in (1, y/b], \) we have:

\[
T(x) \leq T(ax) + T(bx) + x^d
\]

Since \( ax \leq ay/b \leq y \) and \( bx \leq by/b = y \), from the assumption we know:

\[
T(x) \leq C(ax)^d + C(bx)^d + x^d = (C(a^d + b^d) + 1) x^d
\]

From the way we choose the constant \( C \), we know that \( C \geq \beta \). Notice that this means \( C \geq 1/(1 - a^d - b^d) \), so \( C(a^d + b^d) + 1 \leq C \). From this we have:

\[
T(x) \leq Cx^d
\]

Therefore, we can conclude that \( \exists C > 0 \) \( \forall x > 1 T(x) \leq Cx^d \), so \( T(x) = O(x^d) \).

(c) Proof by induction:

We want to prove that if \( r > d \), then \( \exists C, C' > 0 \) \( \forall x \geq 0 \) \( T(x) \leq Cx^r - C'x^d \)

Base case:

For \( 1 < x \leq 1/b \), \( ax \leq bx \leq 1 \) and \( T(x) \leq T(ax) + T(bx) + x^d \leq 1 + 1 + 1/b^d \leq 2 + 1/b^d \).

Let us choose \( C' = 1/(a^d + b^d - 1) \), and \( C = 2 + (1 + C')/b^d \). Because \( d < r \), we have \( a^d + b^d > a^r + b^r = 1 \), and thus \( C' > 0 \). Also, \( C > 2 \). So both \( C \) and \( C' \) are positive numbers. Then:

\[
T(x) \leq 2 + 1/b^d = 2 + (1 + C')/b^d = C - C'/b^d
\]

In the given range of \( x \), \( x^d \geq 1 \) and \( x^d \leq 1/b^d \). So:

\[
T(x) \leq C - C'(1/b^d) \leq Cx^d - C'x^d
\]

This shows that:

\[
\forall x \in (1, 1/b], T(x) \leq Cx^d - C'x^d
\]

Inductive steps:

Assume for some \( y > 1/b \), \( \forall x \in (1, y], T(x) \leq Cx^r - C'x^d \),

Let us consider \( \forall x \in (1, y/b], \) we have:

\[
T(x) \leq T(ax) + T(bx) + x^d
\]

Since \( ax \leq ay/b \leq y \) and \( bx \leq by/b = y \), from the assumption we know:

\[
T(x) \leq C(ax)^r - C'(ax)^d + C(bx)^d - C'(bx)^d + x^d
\]

\[
= C(a^r + b^r) x^r - C'(a^d + b^d - 1/C')x^d = C \cdot 1 \cdot x^r - C' \cdot 1 \cdot x^d
\]

Therefore, we proved that \( \exists C, C' > 0 \) \( \forall x > 1 T(x) \leq Cx^r - C'x^d \). Since \( r > d \), we can conclude that \( T(x) = O(x^r) \).