1. Since the rate of decrease at each site is different, we need to check all possibilities of spending available battery time.

At each site, we calculate the number of pearls collect after each time unit. \( P[i, j], 1 \leq i \leq n, 1 \leq j \leq m \) represents the total pearls collected when the robot is at site \( i \) and the battery time used is \( j \) (\( n \) is number of sites and \( m \) is number of hours available).

The base values, \( P[1, j] \) can be calculated in \( O(m) \) time. \( p[i, j] \) represents the number of pearls collected if robot spends \( j \) units of time at site \( i \). All \( p[i, j] \) values can be calculated in \( O(nm) \) time.

The recursive equation is as follows:

\[
T[i, j] = \max_{t_{i-1} \leq k \leq j} \{ T[i-1, j-k] + p[i, k-t_{i-1}, i] \}
\]

\( i \) varies from 1 to \( n \), \( j \) varies from 1 to \( m \) and \( k \) also varies from 1 to \( m \) (since it depends on \( j \)).

So the runtime complexity for this algorithm is \( O(nm^2) \).

2. We maintain two arrays to solve this problem. First array \( P[i][j] \) is a boolean array which tells whether the substring \( a_i \ldots a_j \) is a palindrome or not. If \( P[i][j] \) is true then the substring \( a_i \ldots a_j \) is a palindrome. Second array \( Cut[j] \) stores minimum number of cuts in best partition for the substring \( a_1 \ldots a_j \).

We can construct the array \( P[i][j] \) using following recurrence

\[
P[i][j] = true \text{ , if } i == j \text{ or } \]
\[
P[i][j] = true, \text{ if } i == j - 1 \text{ and } a_i == a_j \text{ or } \]
\[
P[i][j] = (a_i == a_j) \text{ and } P[i+1][j-1].
\]

And, \( Cut[j] \) can be constructed as,

\[
Cut[j] = 0 \text{ if } P[1][j] = true \text{ or } \]
\[
Cut[j] = \min_{k=0}^{j-1} \{ Cut[k] + 1 \text{ if } P[k+1][j] == true \}
\]

Both the tables can be constructed in \( O(n^2) \) time.
3. You start with leaf nodes, to root. Take a subtree S, with root node $r_S$.

$N_S$ be total nodes in S.

construct an array $conv_S[1..N_S - 1]$

$conv_S[i]$ stores the maximum conviviality value when i nodes from S are in different lists (invited and uninvited) than $r_S$, the root.

If S is leaf, $conv_S[0] = 0$.

Now consider two trees, S1 with root $r_{S1}$ and S2 with root $r_{S2}$. Assume you already have $conv_{S1}$ and $conv_{S2}$. Let S1 has $n_{S1}$ nodes and S2 has $n_{S2}$ nodes.

Now form a new tree S, with $r_{S1}$ as root and $r_{S2}$ as its child. Calculate $conv_S[]$.

Consider calculating $conv_S[k]$. To get exactly k nodes of S in a different list than $r_{S1}$, i nodes from S1 and $k - i$ nodes from S2 are not in same list as $r_{S1}$, for some $0 \leq i \leq k$.

There are two possibilities here: both $r_{S1}$ and $r_{S2}$ are in the same list or they are not. If they are, then conviviality value is $conv_{S1}[i] + conv_{S2}[k - i]$. If they are not, we want $n_{S2} - (k - i)$ nodes to be in different list than $r_{S2}$. Which also means $k - i$ nodes in other list than that of $r_{S1}$. Now, the total conv. value is $conv_{S1}[i] + conv_{S2}[n_{S2} - (k - i)] + conv(r_{S2})$.

$conv_S[k]$ is the maximum of $conv_{S1}[i] + conv_{S2}[k - i]$ and $conv_{S1}[i] + conv_{S2}[n_{S2} - (k - i)] + conv(r_{S2})$ over all i, for which those quantities are defined.

For computing optimal value for a tree with root $r_S$, we first compute the $conv_S$ array for subtrees rooted at $r_S$. Then we can add each child of $r_S$ one at a time. In the end, the optimal value is $conv_S[N_S/2]$.

4. Let $G(V, E)$ be the given graph. $S = \{s_1, s_2, ..., s_i\}$ be the sources and $T = \{t_1, t_2, ..., t_j\}$ be the sinks. We create a new graph $G'$ using $G$ and $S, T$ such that, when we run the Edmonds-Karp on that new graph, we’ll get the max-flow for multisource-multisink problem.

Construct $G' = (V', E')$ as follows:

Add two new vertices $s_0$ as a super-source and $t_0$ as a super-sink.

$$V' = V \cup \{s_0\} \cup \{t_0\}$$

We’ll add extra edges from $s_0$ to all sources $s_i$, with capacity equal to $p_i$. Similarly add extra edges from all sinks $t_i$ to $t_0$, with capacity equal to $q_i$.

Edmonds-Karp on this new graph will result in same max flow as that possible in the multisource-multisink graph $G$.

5. (a) Find a maximum integral s–t flow $f$ using the max flow algorithm. When the algorithm terminates grow an out-tree from s using arcs in the residual graph $G(f)$. This tree does not contain $t$, otherwise we can increase the flow. The vertices $S$ in the tree form a cut for which every arc in $\delta^+(S)$ is at full capacity and every arc in $\delta^+(S)$ has zero flow. The capacity of this cut equals the value of the flow and hence is a minimum cut.
(b) Modify to capacity of each arc to be \( u'_a = mu_a + 1 \) and find a minimum cut with respect to the new capacities. Observe that \( u'(\delta^+(S)) = mu'(\delta^+(S)) + |\delta^+(S)| \) for any \( s-t \) cut \( S \). Since \( 1 \leq |\delta^+(S)| \leq m \) for any cut, it follows easily that the minimum cut wrt \( u' \) is a minimum cut wrt \( u \) containing the fewest possible edges.

(c) Again find a maximum integral \( s-t \) flow \( f \) using the max flow algorithm. Grow an out-tree \( T \) from \( s \) and an in-tree \( T' \) from \( t \) using arcs in the residual graph \( G(f) \). It is not hard to show that every minimum \( s-t \) cut \( S \) has \( V(T) \subseteq S \) and \( S \subseteq V - V(T') \). Hence the minimum cut is unique if and only if \( V(T) = V - V(T') \).

6. (a) i. 2-SAT problem is in \( P \).

It can be solved efficiently by constructing a suitable graph from the given instance. Let us assume \( I \) is the instance of the given clauses. Construct a directed graph \( G \) with -

Vertices of \( G \) will be all the variables \( x_i \) in \( I \).

For each clause \( (x_i \lor x_j) \) put two edges, namely \( (\neg x_i \rightarrow x_j) \) and \( (\neg x_j \rightarrow x_i) \).

If \( I \) has \( n \) variables and \( m \) clauses, then \( G \) has \( 2n \) vertices and \( 2m \) edges, so, size of \( G \) is bounded by a constant multiple of the size of the formula.

A clause \( (x \lor y) \) is equivalent to each of \( \neg x_i \Rightarrow x_j \) and (its contrapositive) \( \neg x_j \Rightarrow x_i \). Thus a formula in 2-SAT may be viewed as a set of implications.

For some variable \( x_i \) in \( I \), if there is a sequence of implications \( x_i \Rightarrow \ldots \Rightarrow \neg x_i \) as well as \( \neg x_i \Rightarrow \ldots \Rightarrow x_i \), then the given 2-SAT formula is unsatisfiable. On the other hand, if implications of the form \( x_i \Rightarrow \ldots \Rightarrow \neg x_i \) and \( \neg x_i \Rightarrow \ldots \Rightarrow x_i \), do not both exist for any variable \( x_i \), then the given formula is satisfiable.

So it is equivalent to determine whether or not directed paths from a vertex \( x_i \) to the vertex \( \neg x_i \) and from the vertex \( \neg x_i \) to \( x_i \) both exist \( G \) for any variable \( x_i \). By creating strongly connected component \( G \), we can check the previous condition in linear time - \( O(|V| + |E|) \). So the computation is linear in the size of formula, so the 2-SAT problem can be solved in polynomial time in the size of its input.

(b) Proof by induction.

Base Case: 3-SAT is shown to be NP-complete.

Inductive Case: Assuming K-SAT is NP-complete, we will show (K+1)-SAT is also NP-complete.

First, note that (K + 1)-SAT is in NP since a non-deterministically generated truth assignment can be evaluated in polynomial time. Now we will show that (K + 1)-SAT is NP-hard by reducing K-SAT to it, completing the proof of NP-completeness.

K-SAT \( \leq_P \) (K + 1)-SAT

1. Let \( F \) be the function which carries out the reduction. We define \( F \) as follows: given an expression

\[
X = A_1 \land \ldots A_n
\]

where each \( A_i \) is a k-term disjunction,

\[
F(X) = (x_{k+1} \lor A_1) \land (x'_{k+1} \lor A_1) \land \ldots \land (x_{k+1} \lor A_n) \land (x'_{k+1} \lor A_n)
\]
2. F creates two disjunctions for every term in \( X \) so it is computable in time proportional to \( 2|X| \).

3. For any boolean expression \( Z \) and any truth assignment of \( x_1 \),

\[
(x \lor Z) \land (x' \lor Z) = (T \lor Z) \land (F \lor Z) = T \land Z = Z
\]

Thus \( F(X) \) is satisfiable if and only if \( X \) is satisfiable.

7. (1 page) From Textbook

in \( \text{NP} \)

Given a potential solution \( x \), it takes polynomial time to compute \( Ax \) and compare it with \( b \). So, this problem is in \( \text{NP} \).

is \( \text{NP-hard} \)

Show: 3-CNF \( \leq_p \) 0-1 Integer Programming.

**Mapping function**

Convert each clause to a row in the matrix. If the clause is \((x_1 \lor \neg x_2 \lor x_3)\), create the matrix row as \(x_1 + (1 - x_2) + x_3 \geq 1\). That is, the variable \( x_1 \) in the clause corresponds to a \( x_1 \) integer programming variable and the negation of \( x_1 \) corresponds to a \( 1 - x_1 \).

And, if we have a solution \( x \) for the mapped 0-1 integer programming problem, the same \( x \) gives the assignment for variables in 3-CNF.

Clearly mapping in both directions is polynomial time.

**Correctness of the Mapping function**

If there is a valid assignments to 3-CNF problem, then inequality should be satisfied for each of the matrix rows with at-least one valid \( x \) (based on the assignments for 3-CNF). If there is no satisfiable assignment, then there should be at-least one clause that fails for any assignment. So, the corresponding inequality will fail for the 0-1 integer problem.

8. in \( \text{NP} \)

Similar to SAT. We just need to do similar verification twice.

is \( \text{NP-hard} \)

Show: 3-CNF \( \leq_p \) Double-SAT

**Mapping**

Given a 3-CNF problem, create a new problem by adding \((x \lor x')\), where \( x \) is a new variable not in original 3-CNF function. If new problem is satisfiable for Double-SAT, then that 3-CNF is satisfiable, otherwise no. The mapping process is clearly polynomial.

**Correctness of the Mapping function**

Now if original 3-CNF has at-least 1 solution, the new function has at-least two solutions. One for which \( x = 1 \) and another for which \( x' = 1 \).
If 3-CNF has no solution, it is impossible to find a solution for Double-SAT.