1. (a) (13 points) Show that an otherwise polynomial-time algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time;

**Proof:** Assume that for $1 \leq i \leq c$ our algorithm calls a subroutine $S_i$ with input $x_i$ of binary length $m_i$. Assume that $S_i$ performs at most $n^{e_i}$ binary operations on inputs of length $n$, where $e_i$ doesn’t depend on $n$. Let $n^{e_0}$ is an upper bound for the running time for our algorithm on input of length $n$ – without the subroutine calls. Then

$$n^{e_0} + m_1^{e_1} + \ldots + m_c^{e_c}$$

is the total running time of the algorithm (with subroutine calls) on an input of length $n$. Here $c, e_0, \ldots, e_c$ do not depend on $n$. By induction we will prove that the input size $m_i$ of $S_i$ is at most $n^{f_i}$ for $f_i \leq \prod_{j=0}^{i-1} e_j$ – not depending on $n$. For $i = 1$ we have $f_1 \leq e_0$, which proves the base case. Assume that the $m_{i-1} \leq n^{f_{i-1}}$. Then the output size of $S_{i-1}$ is at most $(n^{f_{i-1}})^{e_{i-1}}$, therefore the input size of $S_i$ is at most $(n^{f_{i-1}})^{e_{i-1}} = n^{f_i (e_{i-1})}$. (We can assume without loss of generality that the output of $S_{i-1}$ is the input of $S_i$.) Therefore, $f_i \leq \prod_{j=0}^{i-1} e_j$, which proves the inductive step. Thus we have that the total running time is at most

$$n^{e_0} + (n^{f_1})^{e_1} + \ldots + (n^{f_c})^{e_c} \in O\left(n^{\max_{j=0}^{c} e_j} \right)$$

and the exponent of the right hand side does not depend on $n$.

(b) (12 points) Show that a polynomial number of calls to polynomial time subroutines may result in an exponential-time algorithm.

**Proof:** Here is an example for such algorithm: Each subroutine $S_i$ has running time at most $n^{e_i}$ for inputs of size $n$, but the input size of $S_i$ is $m_i := 2^{i-1} n$. For example $S_i$ squares its input $x$ in linear time. Then $S_n$ has input size $2^{n-1} n$, so its running time is linear in $2^{n-1} n$, which is not polynomial time.

2. (15 points) The **subgraph-isomorphism problem** takes two graphs $G_1$ and $G_2$ and asks whether $G_1$ is isomorphic to a subgraph of $G_2$. (Two graphs are isomorphic, if there is a permutation of the vertices which transform one graph into the other, preserving the edges). Prove that the subgraph isomorphism problem is NP-complete, using NP-complete problems discussed in class.

**Proof:** First we prove that the **subgraph-isomorphism problem** is in NP. The certificate is $(G_1 = (V_1, E_1), G_2 = (V_2, E_2), \phi : V_1 \rightarrow V_2)$. The verifying algorithm checks if $\phi$ is a one-to-one function, and for all $u, v \in V_1$ whether $(u, v) \in E_1$ if and only if $(\phi(u), \phi(v)) \in E_2$. 

1
Secondly, we prove that CLIQUE \( \leq_p \) SUBGRAPH ISOMORPHISM. Let \((G = (V, E), k)\) be an input instance for CLIQUE. Define \(G_1\) to be the complete graph on \(k\) vertices, and \(G_2\) to be the graph \(G\). Then \((G_1, G_2) \in\) SUBGRAPH ISOMORPHISM if and only if \((G, k) \in\) CLIQUE.

3. Textbook, page 40 / 2.10 (i) and (iii)
Let \(R\) be a ring (commutative, with 1) and \(a = \sum_{i=0}^{n} a_i x^i \in R[x]\) of degree \(n\), all \(a_i \in R\). The weight \(w(a)\) of \(a\) is the number of nonzero coefficients of \(a\) besides the leading coefficient:
\[
w(a) : = \#\{0 \leq i < n : a_i \neq 0\}.
\]
Thus \(w(a) \leq \deg(a)\), with equality if and only if all coefficients of \(a\) are nonzero. The sparse representation of \(a\) is a list \((i, a_i)_{i \in I}\), with each \(a_i \in R\) and \(a = \sum_{i \in I} a_i x^i\). Then we can choose \#\(I = w(a) + 1\).

(i) (15 points) Show that two polynomials \(a, b \in R[x]\) of weight \(w(a) = n\) and \(w(b) = m\) can be multiplied in the sparse representation using at most \(2nm + n + m + 1\) arithmetic operations in \(R\).

Solution: We modify the polynomial multiplication algorithm (ALGORITHM 2.3) as follows:

SPARSE POLYNOMIAL MULTIPLICATION
INPUT: \(I, J \subset \mathbb{N}, a = [(i, a_i)]_{i \in I}, b = [(j, b_j)]_{j \in J}\) polynomials given in sparse representation
OUTPUT: \(c = a \cdot b\) given in its sparse representation \([(k, c_k)]_{k \in K}\) with \(K \subset \mathbb{N}\).
1. \(K : = \{\}\)
2. \(\text{while } i \in I \text{ do}\)
3. \(\qquad \text{while } j \in J \text{ do}\)
4. \(\qquad \qquad \text{if } i + j \in K \text{ then}\)
5. \(\qquad \qquad \quad c_{i+j} := c_{i+j} + a_i \cdot b_j\)
6. \(\qquad \text{else}\)
7. \(\qquad \quad K := K \cup \{i + j\}\)
8. \(\qquad \quad c_{i+j} := a_i \cdot b_j\)
9. \(\text{fi}\)
10. \(\text{od}\)
11. \(\text{od}\)
12. \(\text{return } [(k, c_k)]_{k \in K}\)

Assume that \#\(I = m+1, \#J = n+1\). Then the above algorithm requires \((n+1)(m+1)\) multiplications in \(R\). The algorithm also conducts an addition in \(R\) if the exponent \(i + j\) was already in \(K\). Thus the total number of additions in \(R\) is \((n+1)(m+1) - \#K\).
Since \#\(K \geq n + m + 1\) (which is the dense case), we have that the number of additions in the worst case is \((n+1)(m+1) - n + m + 1 = nm\). Thus we have at most \((n+1)(m+1) + nm = 2nm + n + m + 1\) arithmetic operations in \(R\).
(iii) (10 points) Let \( n \geq m \). Show that quotient and remainder on division of a polynomial \( a \in R[x] \) of degree less than \( n \) by \( b \in R[x] \) of degree \( m \), with \( \text{l.c.}(b) \) a unit, can be computed using \( n - m \) divisions in \( R \) and \( w(b) \cdot (n - m) \) multiplications and subtractions in \( R \) each.

**Proof:** The polynomial division algorithm (ALGORITHM 2.5) requires \( n - m \) divisions in \( R \), and \( n - m \) constant multiplication and subtraction of the polynomial \( b \). Since the leading term is always 0 after the subtraction of the constant multiple of \( b \), we do not need to compute that term. Therefore each of these polynomial subtractions and multiplications only need \( w(b) \) operations over \( R \), even though \( b \) has \( w(b) + 1 \) terms. Therefore the algorithm uses \( w(b)(n - m) \) subtractions and \( w(b)(n - m) \) multiplications over \( R \).

4. Textbook, page 60/ 3.25 (ii) (10 points)

**Proof of correctness:** Let \( a, b \in \mathbb{N} \) such that \( a \geq b > 0 \). We will consider the following four cases, corresponding to each line of the algorithm

1. If \( a = b \) then \( \gcd(a, b) = a \).
2. If both \( a \) and \( b \) are even then 2 divides the \( \gcd \) and clearly \( \gcd(a, b) = 2 \gcd(a/2, b/2) \).
3. Assume that exactly one among \( a \) and \( b \) are even, say \( a \). Then 2 does not divide the \( \gcd \) and clearly \( \gcd(a, b) = \gcd(a/2, b) \).
4. Assume that neither \( a \) nor \( b \) are even. Let \( d := \gcd(a, b) \) and \( d' = \gcd((a - b)/2, b) \). Since \( d \) divides \( a \) and \( b \), it also divides \( a - b \). But since \( d \) is odd and \( a - b \) is even, \( d \) must divide \( (a - b)/2 \). This implies that \( d \) divides \( d' \).

On the other hand, \( d' \) divides \( b \) and \( (a - b)/2 \), thus it also divides \( a - b \). This implies that \( d' \) divides both \( a \) and \( b \), so it also divides their \( \gcd \) \( d \). Thus \( d \) and \( d' \) must be equal.

Since this four cases cover all possibilities, the algorithm returns the correct answer.

(iii) (10 points) **Solution:**

An upper bound for the depth of the recursion depth is

\[
\lceil \log_2(a) \rceil + \lceil \log_2(b) \rceil.
\]

Assume that the binary length of \( a \) and \( b \) together is \( n \), i.e. \( n = \lceil \log_2(a) \rceil + \lceil \log_2(b) \rceil \).
Then running time satisfies the following recursive inequality:

\[
T(n) \leq T(n - 1) + c \cdot n.
\]

The solution for this is

\[
T(n) \leq \sum_{i=0}^{n} c \cdot i = c \cdot \frac{(n+1)n}{2} \implies T(n) \in O(n^2).
\]
(iv) (15 points) Solution:

INPUT: $a, b \in \mathbb{N}$ such that $a \geq b > 0$

OUTPUT: $\gcd(a, b), s(a, b), t(a, b) \in \mathbb{Z}^3$ such that $\gcd(a, b) = s(a, b) \cdot a + t(a, b) \cdot b$.

1. if $a = b$ then return $\gcd(a, b) = a$ and $s(a, b) = 1, t(a, b) = 0$.
2. if both $a$ and $b$ are even then return $\gcd(a, b) = 2 \gcd(a/2, b/2), s(a, b) : = s(a/2, b/2), t(a, b) := t(a/2, b/2)$.
3. if exactly one among $a$ and $b$ are even, say $a$, then
4. if $S$ is even then return $\gcd(a, b) = \gcd(a/2, b), s(a, b) := (S \pm b)/2, t(a, b) := (T \mp a)/2$
5. if neither $a$ nor $b$ are even then
6. if $S$ is even then return $\gcd(a, b) = \gcd((a-b)/2, b), s(a, b) := S/2, t(a, b) := T - S/2$
7. else return $s(a, b) := (S \pm b)/2, t(a, b) := T - (S \pm a)/2$