Question 1: Write your name and student number.

Solution: Santa Clause, 007

Question 2: The set cover problem is defined as follows:

\[ \text{SetCover} = \{(B, S_1, S_2, \ldots, S_m, K) : B \text{ is a finite set; } m \text{ is an integer; } \]
\[ S_1, S_2, \ldots, S_m \text{ are sets with } \bigcup_{i=1}^{m} S_i = B; \]
\[ K \text{ is an integer; there exists a subset } I \subseteq \{1, 2, \ldots, m\} \text{ of size } K, \text{ such that } \bigcup_{i \in I} S_i = B \} \]

Prove that the language SetCover is in \textbf{NP}.

Solution: The verification algorithm \( V \) does the following:

- It takes as input
  - a finite set \( B \), a sequence \( S_1, S_2, \ldots, S_m \) of sets such that \( \bigcup_{i=1}^{m} S_i = B \), and an integer \( K \),
  - and a set \( I \) of integers.

- The verification algorithm does the following:
  - Check that \( I \subseteq \{1, 2, \ldots, m\} \).
  - Check that the size of \( I \) is equal to \( K \).
  - Check that \( \bigcup_{i \in I} S_i = B \).
  - If all of these are correct, then it returns YES. Otherwise, it returns NO.

The certificate is of course the set \( I \):

\[ (B, S_1, \ldots, S_m, K) \in \text{SetCover} \iff \text{there exists } I \subseteq \{1, \ldots, m\} \]
\[ \text{such that } |I| = K \text{ and } \bigcup_{i \in I} S_i = B \]
\[ \iff \text{there exists a certificate } I \text{ such that } \]
\[ V(B, S_1, \ldots, S_m, K, I) \text{ returns YES.} \]

The length of \( (B, S_1, \ldots, S_m, K) \) is proportional to

\[ \sum_{i=1}^{m} (1 + |S_i|). \]

Note that if \( S_i \) is empty, then \( 1 + |S_i| = 1 \); in this case, the input has to specify something like “here is an empty set”, which adds a constant to the length of \( (B, S_1, \ldots, S_m, K) \).
The length of the certificate $I$ is at most equal to $m$, which is polynomial in the length of $(B, S_1, \ldots, S_m, K)$, because the length of $(B, S_1, \ldots, S_m, K)$ is at least $m$.

What is the running time of the verification algorithm:

- Checking that $I \subseteq \{1, 2, \ldots, m\}$ can be done in $O(|I|) = O(m)$ time.
- Checking that $|I| = K$ can be done in $O(|I|) = O(m)$ time.
- Checking that $\bigcup_{i \in I} S_i = B$ can be done, using brute force, in time
  \[ O \left( \sum_{i=1}^m |S_i| \cdot |B| \right), \]
  which is
  \[ O \left( \left( \sum_{i=1}^m |S_i| + |B| \right)^2 \right), \]
  which is polynomial in the length of $(B, S_1, \ldots, S_m, K)$. (Of course, there are much faster algorithms using sorting and hash tables and balanced binary search trees, etc.)

This shows that SetCover $\in$ NP.

**Question 3:** The (0-1)-integer programming problem with $K$ ones is defined as follows:

\[ \text{IntProg} = \{(A, K) : \text{A is an integer } n \times m \text{ matrix all of whose entries are in } \{0, 1\}; \text{K is an integer; there exists a binary column vector } x \text{ of length } m \text{ with exactly } K \text{ ones, such that } Ax \geq 1 \text{ (componentwise) } \}, \]

where $1$ denotes the column vector of length $n$, all of whose entries are equal to 1.

Prove that SetCover $\leq_p$ IntProg, i.e., in polynomial time, SetCover can be reduced to IntProg.

**Solution:** We need a function $f$ such that

- $f$ maps $(B, S_1, \ldots, S_m, K)$ to $(A, K')$,
- $(B, S_1, \ldots, S_m, K) \in$ SetCover $\Leftrightarrow (A, K') \in$ IntProg,
- the time to compute $(A, K')$ is polynomial in the length of $(B, S_1, \ldots, S_m, K)$.

Here is how we obtain this function $f$. Let $n = |B|$. The matrix $A$ has $n$ rows and $m$ columns. The rows are indexed by the elements of $B$. The $i$-th column will be the characteristic vector $c_i$ for the set $S_i$: The binary vector $c_i$ has length $n$; there is a 1 at position $j$ if and only if the $j$-th element of $B$ is in $S_i$. We also let $K' = K$. 

2
Using brute force, we can compute the matrix $A$ in time 

$$O \left( \sum_{i=1}^{m} |S_i| \cdot |B| \right),$$

which, as we have already seen, is polynomial in the length of $(B, S_1, \ldots, S_m, K)$.

Consider a binary column vector $x$ of length $m$. Let $I$ be the set of indices $i$ such that the $i$-th component of $x$ is 1. Then $Ax$ is a column vector of length $n$, which is equal to 

$$\sum_{i \in I} c_i.$$ 

Note that $\bigcup_{i \in I} S_i = B$ if and only if every entry in the column vector $Ax$ is at least 1. It follows that the following two conditions are equivalent:

- There exists a subset $I \subseteq \{1, 2, \ldots, m\}$ of size $K$, such that $\bigcup_{i \in I} S_i = B$.
- There exists a binary column vector $x$ of length $m$ with exactly $K$ ones, such that $Ax \geq 1$ (componentwise).

**Question 4:** The subset sum problem is defined as follows:

$$\text{SUBSETSUM} = \{(a_1, a_2, \ldots, a_m, b) : \text{ } m, a_1, a_2, \ldots, a_m, b \text{ are integers and } \exists I \subseteq \{1, 2, \ldots, m\} \text{ such that } \sum_{i \in I} a_i = b \}.$$ 

Assume you have a polynomial-time algorithm $A$ that decides, for any input sequence $(a_1, a_2, \ldots, a_m, b)$, whether or not $(a_1, a_2, \ldots, a_m, b) \in \text{SUBSETSUM}$. Note that this algorithm only returns YES or NO; it does not return anything else.

Design a polynomial-time algorithm $B$ that takes an arbitrary sequence $(a_1, a_2, \ldots, a_m, b)$ as input.

- If $(a_1, a_2, \ldots, a_m, b) \in \text{SUBSETSUM}$, then $B$ returns a subset $I$ of $\{1, 2, \ldots, m\}$ such that $\sum_{i \in I} a_i = b$.
- If $(a_1, a_2, \ldots, a_m, b) \not\in \text{SUBSETSUM}$, then $B$ returns NO.

Your algorithm $B$ may use algorithm $A$ as a black box. As always, justify your answer.

**Solution:** Here is the main approach:

- Assume that algorithm $A$ tells us that $(a_1, a_2, \ldots, a_m, b) \in \text{SUBSETSUM}$. Our task is to compute a subset $I \subseteq \{1, 2, \ldots, m\}$ such that $\sum_{i \in I} a_i = b$.

- We “Ask” algorithm $A$ if $(a_1, a_2, \ldots, a_{m-1}, b)$ is in $\text{SUBSETSUM}$.

  - If the answer is YES, then we know that $I \subseteq \{1, 2, \ldots, m-1\}$. Thus, we recurse with the input $(a_1, a_2, \ldots, a_{m-1}, b)$. 

If the answer is NO, then we know that \( m \) belongs to \( I \). Thus, we recurse with the input \((a_1, a_2, \ldots, a_{m-1}, b - a_m)\) and remember that \( m \) is in the final output \( I \).

Based on this, algorithm \( B \) does the following:

- **Algorithm** \( B(a_1, \ldots, a_m, b) \):
  - Run \( A(a_1, \ldots, a_m, b) \).
  - If \( A \) returns NO, then \( B \) returns NO.
  - If \( A \) returns YES, then \( B \) runs \( C(a_1, \ldots, a_m, b, \emptyset) \), where \( C \) is specified below.

Before we give the pseudocode for algorithm \( C \), let us specify what its input and output parameters are:

- **Algorithm** \( C(a_1, \ldots, a_k, b', I') \):
  - \((a_1, \ldots, a_m, b)\) is a YES-instance for \textsc{SubsetSum}.
  - \((a_1, \ldots, a_k, b')\) is a YES-instance for \textsc{SubsetSum}.
  - \( I' \subseteq \{k+1, k+2, \ldots, n\} \).
  - This algorithm returns a set \( I \subseteq \{1, 2, \ldots, m\} \) such that \( I' \subseteq I \) and \( \sum_{i \in I} a_i = b \).

Algorithm \( C \) does the following:

- **Algorithm** \( C(a_1, \ldots, a_k, b', I') \):
  - If \( k = 1 \): Let \( I = I' \cup \{1\} \). Return \( I \) and terminate.
  - If \( k \geq 2 \):
    - Run \( A(a_1, \ldots, a_{k-1}, b') \).
    - If \( A \) returns YES: Run \( C(a_1, \ldots, a_{k-1}, b', I') \).
    - If \( A \) returns NO: Run \( C(a_1, \ldots, a_{k-1}, b' - a_k, I' \cup \{k\}) \).

What is the running time of algorithm \( B \)? We assumed that algorithm \( A \) has polynomial running time; say \( O(m^c) \), where \( c \) is some constant. It follows from the pseudocode that algorithm \( B \) runs algorithm \( A \) at most \( m \) times. Therefore, the running time of algorithm \( B \) is \( O(m^{c+1}) \).

**Question 5:** The Hamilton cycle problem is defined as follows:

\[
\text{HAMILTONCycle} = \{ G : G \text{ is an undirected graph that has a Hamilton cycle} \}.
\]

Let \( \varphi \) be a Boolean formula in the variables \( x_1, x_2, \ldots, x_n \). We say that \( \varphi \) is in conjunctive normal form (CNF) if it is of the form

\[
\varphi = C_1 \land C_2 \land \ldots \land C_m,
\]
where each $C_i$, $1 \leq i \leq m$, is of the following form:

$$C_i = l_1^i \lor l_2^i \lor \ldots \lor l_k^i.$$  

Each $l_j^i$ is a literal, which is either a variable or the negation of a variable.

The satisfiability problem is defined as follows:

$$\text{SAT} = \{ \varphi : \varphi \text{ is in CNF-form and is satisfiable} \}.$$  

Prove that $\text{HamiltonCycle} \leq_P \text{SAT}$, i.e., in polynomial time, $\text{HamiltonCycle}$ can be reduced to $\text{SAT}$.

**Solution:** We need a function $f$ such that

- $f$ maps a graph $G$ to a Boolean formula $\varphi$ in CNF-form,
- $G$ has a Hamilton cycle $\iff$ $\varphi$ is satisfiable,
- the time to compute $\varphi$ is polynomial in the length of $G$.

Let $G = (V, E)$ be an undirected graph with $V = \{v_1, v_2, \ldots, v_n\}$. Recall that a Hamilton cycle is a permutation $u_1, u_2, \ldots, u_n$ of $V$ such that $\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_{n-1}, u_n\}, \{u_n, u_1\}$ are edges in $E$. We are going to encode the existence of such a cycle as a Boolean formula.

We will use $n^2$ Boolean variables $x_{ij}$, $1 \leq i \leq n$, $1 \leq j \leq n$. The meaning of these variables is as follows:

$$x_{ij} = true \iff \text{vertex } v_i \text{ is at position } j \text{ in the permutation.}$$

Consider a vertex $v_i$. We note that

$$x_{i1} \lor x_{i2} \lor \ldots \lor x_{in} = true \iff v_i \text{ occurs at least once.}$$

For two indices $j \neq k$, $x_{ij} \land x_{ik} = true$ if and only if $v_i$ is at positions $j$ and $k$. Thus, $x_{ij} \land x_{ik}$ must be false, which is the same as saying that $\neg(x_{ij} \land x_{ik})$ must be true, which is the same as saying that

$$\neg x_{ij} \lor \neg x_{ik}$$

must be true. Thus,

$$\bigwedge_{j \neq k} (\neg x_{ij} \lor \neg x_{ik}) = true \iff v_i \text{ occurs at most once.}$$

Let

$$\varphi_1 = \bigwedge_{i=1}^n \left( (x_{i1} \lor x_{i2} \lor \ldots \lor x_{in}) \land \bigwedge_{j \neq k} (\neg x_{ij} \lor \neg x_{ik}) \right).$$

Then

$$\varphi_1 = true \iff \text{each } v_i \text{ occurs exactly once.}$$
The size of $\varphi_1$ is $O(n^3)$ and it can be constructed in time $O(n^3)$.

We need more to guarantee that we have a permutation of the vertex set. Note that we can make $\varphi_1$ true by placing all vertices at the same position.

Consider a position $j$. We note that

$$x_1j \lor x_2j \lor \ldots \lor x_nj = true \iff \text{there is at least one vertex at position } j.$$ 

Let

$$\varphi_2 = \bigwedge_{j=1}^{n} (x_1j \lor x_2j \lor \ldots \lor x_nj).$$

Then

$$\varphi_2 = true \iff \text{each position has at least one vertex.}$$

The size of $\varphi_2$ is $O(n^2)$ and it can be constructed in time $O(n^2)$.

So far, we have that

$$\varphi_1 \land \varphi_2 = true \iff \text{we have a permutation of the vertex set.}$$

Let $\{v_i, v_i'\}$ not be an edge in the edge set of $G$. We note that

$$(x_{ij} \land x_{i',j+1}) \lor (x_{i',j} \land x_{i,j+1})$$

must be false. Thus,

$$\neg((x_{ij} \land x_{i',j+1}) \lor (x_{i',j} \land x_{i,j+1}))$$

must be true, which is the same as

$$(\neg x_{ij} \lor \neg x_{i',j+1}) \land (\neg x_{i',j} \lor x_{i,j+1}).$$

Let

$$\varphi_3 = \bigwedge_{\{v_i, v_i'\} \not\in E} \bigwedge_{j=1}^{n-1} ((\neg x_{ij} \lor \neg x_{i',j+1}) \land (\neg x_{i',j} \lor x_{i,j+1}))$$

and

$$\varphi_4 = \bigwedge_{\{v_i, v_i'\} \not\in E} ((\neg x_{i,n} \lor \neg x_{i',1}) \land (\neg x_{i',n} \lor x_{i,1})).$$

Then

$$\varphi_3 \land \varphi_4 = true \iff \text{each neighboring pair of vertices is an edge.}$$

The size of $\varphi_3 \land \varphi_4$ is $O(n^3)$ and it can be constructed in time $O(n^3)$.

To conclude, if we let

$$\varphi = \varphi_1 \land \varphi_2 \land \varphi_3 \land \varphi_4,$$

then $\varphi$ is satisfiable if and only if the graph $G$ has a Hamilton cycle. The total time to construct $\varphi$ is $O(n^3)$.

**Question 6:** After being successful in implementing a superfast sorting algorithm, Lionel Messi decides to continue working as a software developer. Lionel looks again at his previous
assignment, where he used Dijkstra’s algorithm to sort a sequence of numbers. He realizes that he showed that the sorting problem can be reduced to one run of Dijkstra’s algorithm. Suddenly, Lionel actually understands the notion of polynomial-time reductions. Because of this, he decides to solve the P versus NP problem. Below, you find the proof of what is now known as

**Messi’s Theorem: P = NP.**

Let $\varphi$ be a Boolean formula in the variables $x_1, x_2, \ldots, x_n$.

We say that $\varphi$ is in *conjunctive normal form* (CNF) if it is of the form

$$\varphi = C_1 \land C_2 \land \ldots \land C_m,$$

where each $C_i$, $1 \leq i \leq m$, is of the following form:

$$C_i = l^i_1 \lor l^i_2 \lor \ldots \lor l^i_{k_i}.$$

Each $l^i_j$ is a *literal*, which is either a variable or the negation of a variable.

We say that $\varphi$ is in *disjunctive normal form* (DNF) if it is of the form

$$\varphi = C_1 \lor C_2 \lor \ldots \lor C_m,$$

where each $C_i$, $1 \leq i \leq m$, is of the following form:

$$C_i = l^i_1 \land l^i_2 \land \ldots \land l^i_{k_i}.$$

Again, each $l^i_j$ is a literal.

We define the following two languages:

$$\text{Sat} = \{ \varphi : \varphi \text{ is in CNF-form and is satisfiable} \}$$

and

$$\text{DNFSAT} = \{ \varphi : \varphi \text{ is in DNF-form and is satisfiable} \}.$$

(6.1) Lionel starts by proving that $\text{DNFSAT} \in \text{P}$. Your task is to present a proof of this fact.

**Solution:** Consider a Boolean formula

$$\varphi = C_1 \lor C_2 \lor \ldots \lor C_m$$

in DNF-form. Thus, each $C_i$, $1 \leq i \leq m$, is of the form

$$C_i = l^i_1 \land l^i_2 \land \ldots \land l^i_{k_i}.$$

We need two observations:

- $\varphi$ is satisfiable if and only if at least one clause $C_i$ is satisfiable.
• The clause $C_i$ is satisfiable if and only if it does not contain a variable, say $x_j$, and its negation $\neg x_j$.

This leads to the following algorithm:

1. For each $i = 1, 2, \ldots, m$:
   (a) For each $j = 1, 2, \ldots, n$: Check if the clause $C_i$ contains both $x_j$ and $\neg x_j$.
   (b) If there is no such $j$: return “input is satisfiable” and terminate the algorithm.

2. Return “input is not satisfiable”.

The running time of the algorithm is

$$O \left( \sum_{i=1}^{m} n \cdot k_i \right).$$

The size of the Boolean formula $\varphi$ is something like

$$n + \sum_{i=1}^{m} k_i.$$

Therefore, the running time is at most quadratic in the size of $\varphi$.

Note that there are faster algorithms to do this. However, a quadratic running time is enough, because we only care that it is polynomial.

(6.2) Here is Lionel’s argument to complete the proof of Messi’s Theorem:

• Let $\varphi$ be an arbitrary Boolean formula in CNF-form. We can use the basic rules of logic (such as De Morgan’s Law) to rewrite $\varphi$ as an equivalent Boolean formula in DNF-form. Therefore,

$$\text{Sat} \leq_P \text{DNFSAT}.$$  

• We have seen in (6.1) that $\text{DNFSAT} \in \text{P}$. 

• Since $\text{Sat} \leq_P \text{DNFSAT}$ and $\text{DNFSAT} \in \text{P}$, we have $\text{Sat} \in \text{P}$.

• Lionel remembers from COMP 3804 that $\text{Sat}$ is $\text{NP}$-complete.

• Thus, the $\text{NP}$-complete problem $\text{Sat}$ belongs to $\text{P}$.

• Therefore, $\text{P} = \text{NP}$.

Is Lionel’s proof of Messi’s Theorem correct? As always, justify your answer.

Solution: As you can guess, Lionel’s proof is wrong. The mistake is in the claim that

$$\text{Sat} \leq_P \text{DNFSAT}.$$
Consider the CNF Boolean formula
\[ \varphi = (x_1 \lor y_1 \lor z_1) \land (x_2 \lor y_2 \lor z_2) \land \cdots \land (x_m \lor y_m \lor z_m) \]
where the variables are \( x_i, y_i, \) and \( z_i, \) for \( i = 1, 2, \ldots, m. \)

In COMP 1805, you learned that
\[ x \lor (y \land z) \]
and
\[ (x \lor y) \land (x \lor z) \]
are logically equivalent. Also
\[ x \land (y \lor z) \]
and
\[ (x \land y) \lor (x \land z) \]
are logically equivalent.

If we use these rules to convert \( \varphi \) to an equivalent DNF formula, we get a formula with \( 3^m \) clauses. Each such clause is of the form
\[ c_1 \land c_2 \land \cdots \land c_m, \]
where \( c_1 \in \{x_1, y_1, z_1\}, c_2 \in \{x_2, y_2, z_2\}, \ldots, c_m \in \{x_m, y_m, z_m\}. \) The size of this DNF formula is proportional to \( 3^m; \) the time to write it down is thus \( \Omega(3^m), \) which is not polynomial in the length of \( \varphi. \)

Sorry Lionel! No million dollars for you.