This problem sheet is about difficult computational problems [1].

1. Explain what is meant by SUBSETSUM, and the decision problem related to it.

> **Solution:** SUBSETSUM is the set of all sets of integers that have a subset the sum of all whose elements is zero.
>
> $$\text{SUBSETSUM} = \{S \mid S \subseteq \mathbb{Z}, \exists T \subseteq S : \sum_{i \in T} i = 0\}$$

2. Write a function in Racket that decides SUBSETSUM.

> **Solution:**

```
(define (subsetsum l)
  (not (null? (filter
    (lambda (y) (= 0 (apply + y)))
    (filter
      (lambda (x) (not (null? x)))
      (combinations l))))))
```

3. Write a function in Java that decides SUBSETSUM.

> **Solution:**

```
// From: www.geeksforgeeks.org/dynamic-programming-subset-sum-problem
static boolean isSubsetSum(int set[], int n, int sum) {
  if (sum == 0)
    return true;
  if (n == 0 && sum != 0)
    return false;

  if (set[n-1] > sum)
    return isSubsetSum(set, n-1, sum);

  return isSubsetSum(set, n-1, sum) ||
           isSubsetSum(set, n-1, sum-set[n-1]);
}
```

4. Explain the integer factorisation problem and why it is important in modern cryptography.

Solution: The integer factorisation problem is the problem of factorising a natural number (greater than one) into a product of prime numbers. Every such number is a distinct product of primes, up to re-organsing the order of multiplication. For instance, the only way to factorise the number 561 into a product of primes is $3 \times 11 \times 17$.

The complexity of integer factorisation is not know. No polynomial time algorithm has been developed, and it is not known if the problem is NP-complete. Modern asymmetric key cryptgraphic systems rely on the factorisation of a product of two similar-sized prime numbers not being efficiently possible. The fact that PRIMES is in P means that we can find two similar sized prime numbers efficiently. Modern cryptography exploits these two facts.

5. Explain what is meant by SAT.

Solution: SAT is the set of all Boolean formulas where some setting of the constituent Boolean variables satisfies the formula. For instance, the formula $a \wedge (b \vee c)$ is satisfiable (set a and b to true), but $a \wedge \neg a$ is not.

6. Explain the 3-SAT decision problem.

Solution: 3-SAT is the set of all satisfiable Boolean formulas in Conjunctive Normal Form where each clause contains three literals. The decision problem related to 3-SAT is the problem of deciding 3-SAT. Sometimes this is called the 3-SAT decision problem.

7. Explain the terms conjunctive normal form and disjunctive normal form.

Solution: A Boolean formula is in Conjunctive Normal Form (CNF) if it is a conjunction of disjunctions. That is, it is a series of clauses with ands between them, where each clauses is a series of literals with ors between them. A literal is a variable or its negation. For example, the following formula is in CNF.

$$(a \vee b \vee negc) \wedge (\neg a \vee c)$$

Disjunctive Normal Form (DNF) is a disjunction of conjunctions – the ors are outside the brackets, and the ands are inside. For example, the following is a formula in DNF.

$$(a \wedge b) \vee (\neg a \wedge \neg b)$$

8. Convert the following expressions to Conjunctive Normal Form.

(a) $a \vee b$
(b) $a \wedge b$
(c) $((a \wedge b) \vee (\neg b \wedge c)) \vee \neg d$
(d) $(a \wedge b) \vee (c \wedge d)$
(e) $(a \vee b) \wedge (c \vee d)$

Solution:

(a) $a \vee b$

(b) $a \wedge b$

(c) $(a \vee \neg b \vee \neg d) \wedge (b \vee c \vee \neg d)$

(d) $(a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$

(e) $(a \vee b) \wedge (c \vee d)$

9. Convert the following expressions to Disjunctive Normal Form.
   (a) $a \vee b$
   (b) $a \wedge b$
   (c) $((a \wedge b) \vee (\neg b \wedge c)) \vee \neg d$
   (d) $(a \wedge b) \vee (c \wedge d)$
   (e) $(a \vee b) \wedge (c \vee d)$

Solution:

(a) $a \vee b$

(b) $a \wedge b$

(c) $(a \wedge b) \vee (\neg b \wedge c) \vee \neg d$

(d) $(a \wedge b) \vee (c \wedge d)$

(e) $(a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d)$

10. Determine if there is a setting of the variables in the following expression that makes the evaluation of the expression true.
   (a) $a \vee b$
   (b) $a \wedge b$
   (c) $((a \wedge b) \vee (\neg b \wedge c)) \vee \neg d$
   (d) $(a \wedge b) \vee (c \wedge d)$

(e) $(a \vee b) \wedge (c \vee d)$

Solution:

(a) $(a, b) = (1, 1)$

(b) $(a, b) = (1, 1)$

(c) $(a, b, c, d) = (1, 1, 1, 0)$

(d) $(a, b, c, d) = (1, 1, 1, 0)$

(e) $(a, b, c, d) = (1, 1, 1, 0)$

11. Explain how to prove that a decision problem is NP-complete.

Solution: A decision problem is NP-complete if it is noth NP-hard and in NP itself. Thus, to show a decision problem is NP-complete we must show that it satisfies both of these properties. A decision problem is in NP if an instance can be verified in polynomial time, which is the same as being decidable in polynomial time by a non-determinisitic Turing machine.

A problem is NP-hard if every problem in NP can be reduced to it in polynomial time. A reduction from problem B to problem A is an algorithm for converting any instance of B to an instance of A. The easiest way to show a problem in NP is NP-complete is to reduce an NP-complete problem to it in polynomial time to .

12. Prove that 3-SAT is NP-complete. You may assume that SAT is NP-complete.

Solution: The set 3-SAT is a subset of SAT, and SAT is in NP. Thus, 3-SAT is in NP.

We can convert an instance of SAT to an instance of 3-SAT in the following way. First convert the formula to CNF. Then convert any one-literal clause a of the SAT formula to the following 3-SAT formula:

$$(a \vee u_1 \vee u_2) \wedge (a \vee u_1 \vee \neg u_2) \wedge (a \vee \neg u_1 \vee u_2) \wedge (a \vee \neg u_1 \vee \neg u_2).$$

Convert any two-variable clause $a \vee b$ to:

$$(a \vee b \vee u_1) \wedge (a \vee b \vee \neg u_1)).$$

Any three-variable clause can be left as-is. Convert any clause with more than three variables $a \vee b \vee c \vee \ldots$ to:

$$(a \vee b \vee u_1) \wedge (c \vee \neg u_1 \vee u_2)) \wedge \ldots \wedge (i \vee \neg u_{n-4} \vee u_{n-3}) \wedge (j \vee k \vee \neg u_{n-3}).$$

Now the formula is in 3-CNF. This reduction is polynomial in time, because the output is at most three times as long as the input.

References

[1] Micharl Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 3rd edition, 1996.