This problem sheet is about computational complexity [1].

1. Determine the number of comparisons made by Bubble sort on the following inputs.
   (a) $[3, 2, 1]$
   (b) $[4, 3, 2, 1]$
   (c) $[5, 4, 3, 2, 1]$
   (d) $[6, 5, 4, 3, 2, 1]$
   (e) $[20, 19, 18, \ldots, 3, 2, 1]$
   (f) $[3, 4, 5, 2, 1]$
   (g) $[4, 5, 1, 2, 3]$

Solution:

(a) $2 + 1 = 3$

(b) $3 + 2 + 1 = 6$

(c) $4 + 3 + 2 + 1 = 10$

(d) $5 + 4 + 3 + 2 + 1 = 15$

(e) $6 + 4 + 3 + 2 + 1 = 21$

(f) $(20 \times 19) \div 2 = 190$

(g) $4 + 3 + 2 + 1 = 10$

(h) $5 + 4 + 3 = 12$, using the fact that if no swaps are made in one pass then the list is sorted.

2. Classify the following as polynomial, exponential, or logarithmic expressions.
   (a) $3n + 1$
   (b) $n^2 + 2n + 1$
   (c) $log_b(a)$
   (d) 10^n
   (e) $2^n + n^2$
   (f) $nlog_n$
   (g) n^n

Solution:

(a) Polynomial (linear).

(b) Polynomial.

(c) Logarithmic.

(d) Exponential.

(e) Ambiguous. The exponential term 2^n will grow much quicker than the polynomial term n^2.

(f) Ambiguous. Sometimes called linearithmic.

(g) Exponential.

3. Explain what the P computational complexity class is, and give an example of a problem known to be in P.

Solution: An algorithm is said to be solvable in *polynomial time* if the number of steps required to complete the algorithm for a given input is $O(n^k)$ for some nonnegative integer k, where n is the size of the algorithm's input.

In 2002, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena proved, in a paper called *PRIMES is in P*, that the problem of determining whether a given natural number was a prime or not is in P. Their algorithm is often referred to as the AKS primality test.

4. Explain what PRIMES is.

Solution:

$$\text{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$$

PRIMES is the set of all prime numbers, and is a subset of the natural numbers.

5. Describe two different algorithms the check if a number is a prime. The algorithms should accept a single positive integer as input, and output true if the number is prime and false otherwise.

Solution:

```java
public static boolean is_prime_brute(int n) {
  for (int i = 2; i < n; i++)
    if (n % i == 0)
      return false;
  return true;
}

public static boolean is_prime_sqrt(int n) {
  for (int i = 2; i < Math.sqrt(n); i++)
    if (n % i == 0)
      return false;
  return true;
}
```

6. Determine which of the following are in PRIMES (without Google).
   (a) 2
   (b) 3
   (c) 4
   (d) 10
   (e) 11
   (f) 13,109
   (g) 100,827
   (h) 102,203

Solution:

(a) Yes

(b) Yes

(c) No

(d) No

(e) Yes

(f) Yes

(g) No

(h) Yes

7. Explain what a decision problem is, and how decision problems relate to Turing machines.

Solution: A decision problem is a problem where the answer takes on one of two values, typically true or false. For convenience, we usually represent true by 1 and false by 0. We also usually encode instances of the problem in binary. In that case, a decision problem f has the following form.

$$f : \{0,1\}^n \to \{0,1\}$$

Decision problems are studied extensively in the context of Turing machines. For a given input, a given Turing machine can end up in three different scenarios. It can end in the accept state, it can end in the fail state, or it can end up in an infinite loop. Turing machines that never end in an infinite loop are said to decide the language they accept. In this context, accept represents an outcome of 1, and fail represents an outcome of 0 in the notation above.

8. Explain the decision problem related to PRIMES.

Solution: PRIMES is the set of all prime numbers.

$$\text{PRIMES} = \{i \mid i \in \mathbb{N}, i \text{ is prime}\}$$

The decision problem related to primes is the function f where:

$$f(i) : \mathbb{N} \to \{0,1\} : \begin{cases} 1 & \text{if } i \text{ is prime} \\ 0 & \text{otherwise} \end{cases}$$

So, the decision problem takes as input any natural number. It outputs 1 if that number is a prime and 0 otherwise.

9. Explain the concept of complexity in terms of Turing machines.

Solution: A Turing machine essentially consists of a (linear) tape divided into cells, and a state table. Each cell contains a single symbol. At any given point in time the machine is in a single state, and is reading the symbol in a single cell. The state table gives the rules as to what to do for the current symbol being read and the current state the machine is in. The state table will indicate a symbol to overwrite the symbol in the current cell, the neighbouring cell to which the machine should move next, and the next state the machine should enter.

Let n be the number of cells on the tape that do not initially (before the machine runs) contain the blank symbol. Call this the size of input. Now consider the language that the Turing machine accepts, and call it L. This is the set of all possible inputs to the Turing machine in which it ends in the accept state.

For each element of L, consider the number of times we have to look up the state table before the initial input is accepted. Pick the worst of these, and call that the *worst case*. In general, L is an infinite set, so we must express the worst case in terms of n. Let $f(n)$ be the function of n expressing the number of look-ups in the worst case. The complexity of the Turing machine is $O(f(n))$.

10. Explain why if we can solve decision problem A in polynomial time, and we can convert decision problem B to problem A in polynomial time, then we can solve problem B in polynomial time too.

Solution: Let $f(m)$ be the number of steps it takes in the worst case to solve an input of length m for A. Let $g(n)$ be the number of steps it takes in the worst case to convert an instance of B to A. Then in the worst case, an algorithm that converts B to A and then solves A takes at most $f(m) + g(n)$ steps to complete. So, the algorithm is $O(f(m) + g(n))$.

Now, we know that f and g are polynomials, because we were told they are in the question. However, f is a polynomial in m and g in n. Can we express n in terms of a polynomial in m? We must be able to, because a Turing machine that works in polynomial time will always halt with a tape output that is of length a polynomial in the length of the input.

References

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