1. An *algorithm* is a finite set of precise instructions for performing a computation.

2. Properties that algorithms generally share:
   Input, Output, Definiteness, Correctness, Effectiveness, Generality.

3. Fundamental constructs: if-then; while; for loops.

4. Example: FIND-MAX (Algorithm 1).

5. Two Search Algorithms:

   (a) Linear search (Algorithm 2).

   (b) Binary search (Algorithm 3).

6. Two Sorting Algorithms:

   (a) Bubble sort (Algorithm 4).

   (b) Insertion sort (Algorithm 5).

7. Greedy algorithms (make the best choice at each step with only knowledge available).

8. Complexity of algorithms: measured in space (memory) and time requirements.
   Time complexity is expressed in terms of the number of operations used by the algorithm when the input has a particular size.

   (a) Time complexity of LINEAR-SEARCH:
   If $x = a_i$ for some i, then $2i + 1$ comparisons. Therefore, the most comparisons are $2n + 2$ ($= 2n$ (in the loop) $+ 1$ (exit loop) $+ 1$ (outside loop)), are required when the element is not in the list.

   (b) Time complexity of BINARY-SEARCH:
   Assume that there are $n = 2^k$ elements in the list. At most $2k + 1 + 1 = 2 \log n + 2$ comparisons are required where $2k + 1 + 1 = 2 \times$ (# of loops) $+ 1$ (exit loop) $+ 1$ (outside loop).

   (c) Time complexity of BUBBLE-SORT:
   The total number of comparisons is $(n - 1) + (n - 2) + \cdots + 2 + 1 = \frac{(n-1)n}{2}$.

   (d) Time complexity of INSERTION-SORT:
   The total number of comparisons is $1 + 2 + \cdots + (n - 1) = \frac{(n-1)n}{2}$.

Algorithm 1 An algorithm for finding the maximum element in a finite sequence $a = \{a_1, \ldots, a_n\}$ of integers.

```
FIND-MAX (a)
  max ← a_1
  for i ← 2, 3, ..., n do
    if max < a_i then
      max ← a_i
    end if
  end for
  return max {max is the largest element.}
```

Algorithm 2 An algorithm for locating an element x in a list of distinct elements $a = \{a_1, \ldots, a_n\}$ of integers, or determine that it is not in the list.

```
LINEAR-SEARCH (a, x)
  i ← 1
  while i ≤ n and x ≠ a_i do
    i ← i + 1
  end while
  if i ≤ n then
    location ← i
  else
    location ← 0
  end if
  return location {location is the subscript of the term that equals x, or is 0 if x is not found.}
```

Algorithm 3 An algorithm for locating an element x in a list of integers $a = \{a_1, \ldots, a_n\}$ occurring in an increasing order, or determine that it is not in the list.

```
BINARY-SEARCH (a, x)
  i ← 1 {i is the left endpoint of search interval}
  j ← n {j is the right endpoint of search interval}
  while i < j do
    m ← ⌊(i + j)/2⌋
    if x > a_m then
      i ← m + 1
    else
      j ← m
    end if
  end while
  if x = a_i then
    location ← i
  else
    location ← 0
  end if
  return location {location is the subscript of the term that equals x, or is 0 if x is not found}
```

Algorithm 4 An algorithm for sorting a list of integers $a = \{a_1, \ldots, a_n\}$ into a list in which the elements are in increasing order.

```
BUBBLE-SORT (a)
  for i ← 1, ..., n − 1 do
    for j ← 1, ... n − i do
      if a_j > a_{j+1} then
        temp ← a_j
        a_j ← a_{j+1}
        a_{j+1} ← temp
      end if
    end for
  end for
  return a {a_1, a_2, ..., a_n is in increasing order}
```

Algorithm 5 An algorithm for sorting a list of integers $a = \{a_1, \ldots, a_n\}$ into a list in which the elements are in increasing order.

```
INSERTION-SORT (a)
  for j ← 2, ..., n do
    i ← 1
    while a_j > a_i do
      i ← i + 1
    end while
    m ← a_j
    for k ← 0, ..., j − i − 1 do
      a_{j−k} ← a_{j−k−1}
    end for
    a_i ← m
  end for
  return a {a_1, a_2, ..., a_n is in increasing order}
```

The HALTING Problem, 1936 Alan Turing. One of the most famous theorems in computer science. There exist problems that cannot be solved using any procedure (i.e. there exist unsolvable problems). There does not exist a procedure which given a program P and input to the program I determines whether program P completes (halts) given input I or runs forever.