Handout 2.5 & 3.1

Definition: The sets A and B have the same *cardinality* if and only if there is a one-to-one correspondence from A to B. When A and B have the same cardinality, we write |A| = |B|.

Definition: If there is a one-to-one function from A to B, the cardinality of A is less than or the same as the cardinality of B and we write |A| ≤ |B|. Moreover, when |A| ≤ |B| and A and B have different cardinality, we say that the cardinality of A is less than the cardinality of B and we write |A| < |B|.

Definition: A set that is either finite or has the same cardinality as the set of positive integers is called *countable*. A set that is not countable is called *uncountable*. When an infinite set S is countable, we denote the cardinality of S by

$$\aleph_0$$

where $\aleph$ is aleph, the first letter of the Hebrew alphabet). We write

$$|S| = \aleph_0$$

and say that S has cardinality "aleph null."

Theorem: If A and B are countable sets, then A ∪ B is also countable.

SCHRÖDER-BERNSTEIN THEOREM If A and B are sets with |A| ≤ |B| and |B| ≤ |A|, then |A| = |B|. In other words, if there are one-to-one functions f from A to B and g from B to A, then there is a one-to-one correspondence between A and B.

Definition: We say that a function is **computable** if there is a computer program in some programming language that finds the values of this function. If a function is not computable we say it is **uncomputable**.

Definition: An *algorithm* is a finite sequence of precise instructions for performing a computation or for solving a problem.

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max {max is the largest element}

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** $location := i$
else $location := 0$
return $location$ {$location$ is the subscript of the term that equals x, or is 0 if x is not found}

ALGORITHM 3 The Binary Search Algorithm.

procedure *binary search* (x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is left endpoint of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ **then** $i := m + 1$
    else $j := m$
if $x = a_i$ **then** $location := i$
else $location := 0$
return $location$ {$location$ is the subscript i of the term a_i equal to x, or 0 if x is not found}

ALGORITHM 5 The Insertion Sort.

procedure *insertion sort*($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i + 1$
    $m := a_j$
    for $k := 0$ **to** $j - i - 1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, \ldots, a_n$ is in increasing order}

Lemma: If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change in dimes, nickels, and pennies cannot exceed 24 cents.

Theorem: The greedy algorithm (Algorithm 6) produces change using the fewest coins possible.

ALGORITHM 7 Greedy Algorithm for Scheduling Talks.

procedure *schedule*($s_1 \leq s_2 \leq \cdots \leq s_n$: start times of talks,
$e_1 \leq e_2 \leq \cdots \leq e_n$: ending times of talks)
sort talks by finish time and reorder so that $e_1 \leq e_2 \leq \ldots \leq e_n$
$S := \emptyset$
for $j := 1$ **to** n
  if talk j is compatible with S **then**
    $S := S \cup \{\text{talk } j\}$
return S {S is the set of talks scheduled}

ALGORITHM 6 Greedy Change-Making Algorithm.

procedure *change*($c_1, c_2, \ldots, c_r$: values of denominations of coins, where
$c_1 > c_2 > \cdots > c_r$; n: a positive integer)
for $i := 1$ **to** r
  $d_i := 0$ {d_i counts the coins of denomination c_i used}
  while $n \geq c_i$
    $d_i := d_i + 1$ {add a coin of denomination c_i}
    $n := n - c_i$
{d_i is the number of coins of denomination c_i in the change for $i = 1, 2, \ldots, r$}

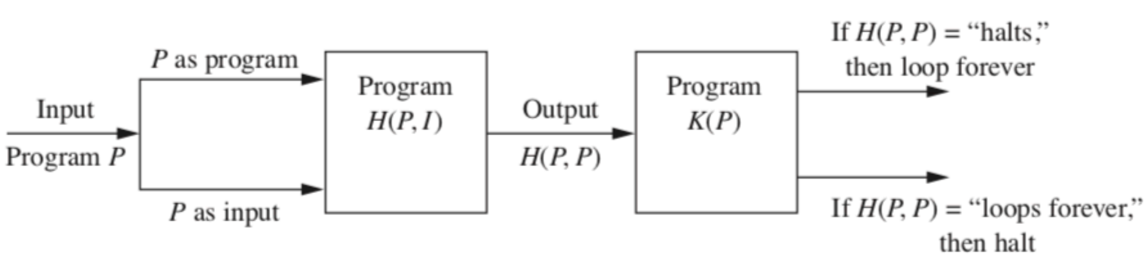

FIGURE 2 **Showing that the Halting Problem is Unsolvable.**