ชื่อ-สกุล ______________________________ รหัสนักศึกษา ____________________

ซูโดโค้ดของอัลกอริทึมต่าง ๆ (Pseudocodes)

Dynamic Programming

Algorithm 2 CoinRow($C[1 \ldots n]$)

```
Require: $C[1 \ldots n]$, an array of positive integers indicating the coin values
Ensure: The maximum amount of money that can be picked up
  $F[0] \leftarrow 0$
  $F[1] \leftarrow C[1]$
  for $i \leftarrow 2$ to $n$ do
      $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
  end for
  Return $F[n]$
```

Algorithm 3 ChangeMaking($D[1 \ldots m], n$)

```
  Applies dynamic programming to find the minimum number of coins of denominations $d_1 < d_2 < \cdots < d_m$ where $d_1 = 1$ that add up to a given amount $n$
Require: Positive integer $n$ and array $D[1 \ldots m]$ of increasing positive integers indicating the coin dominations where $D[1] = 1$
Ensure: The minimum number of coins that add up to $n$
  $F[0] \leftarrow 0$
  for $i \leftarrow 1$ to $n$ do
      $temp \leftarrow \infty$
      $j \leftarrow 1$
      while $j \leq m$ And $i \geq D[j]$ do
          $temp \leftarrow \min(F[i - D[j]], temp)$
          $j \leftarrow j + 1$
      end while
      $F[i] \leftarrow temp + 1$
  end for
  Return $F[n]$
```

Algorithm 4 RobotCoinCollection($C[1 \ldots n, 1 \ldots m]$)

```
  Applies dynamic programming to compute the largest number of coins a robot can collect on a $n \times m$ board by starting at (1,1) and moving right and down from upper left to down right corner
Require: Matrix $C[1 \ldots n, 1 \ldots m]$ whose elements are equal to 1 and 0 for cells with and without a coin, respectively.
Ensure: Largest number of coins the robot can bring to cell $(n, m)$
  $F[1,1] \leftarrow C[1,1]$
  for $j \leftarrow 2$ to $m$ do
      $F[1,j] \leftarrow F[1,j-1] + C[1,j]$
  end for
  for $i \leftarrow 2$ to $n$ do
      $F[i,1] \leftarrow F[i-1,1] + C[i,1]$
      for $j \leftarrow 2$ to $m$ do
          $F[i,j] \leftarrow \max(F[i-1,j], F[i,j-1]) + C[i,j]$
      end for
  end for
  Return $F[n,m]$
```

Algorithm 5 MFKnapsack(i, j)

```
  *Implements the memory function method for the knapsack problem*
Require: A nonnegative integer $i$ indicating the number of the first items being considered and a nonnegative integer $j$ indicating the knapsack capacity
Ensure: The value of an optimal feasible subset of the first $i$ items
  Note: Uses as global variables input arrays $Weights[1 \ldots n], Values[1 \ldots n]$, and table $F[0 \ldots n, 0 \ldots W]$ whose entries are initialized with $-1$'s except for row 0 and column 0 initialized with 0's
  if $F[i, j] < 0$ then
    if $j < Weights[i]$ then
      $value \leftarrow MFKnapsack(i-1, j)$
    else
      $value \leftarrow \max(MFKnapsack(i-1, j), Values[i] + MFKnapsack(i-1, j - Weights[i]))$
    end if
    $F[i, j] \leftarrow value$
  end if
  Return $F[i, j]$
```

Greedy Techniques

Algorithm 6 Prim(G)

```
Require: A weighted connected graph $G = \langle V, E \rangle$
Ensure: $E_T$, the set of edges composing a minimum spanning tree of $G$
  $V_T \leftarrow \{v_0\}$      ▷ The set of tree vertices can be initialized with any vertex
  $E_T \leftarrow \emptyset$
  for $u \leftarrow 1$ to $|V| - 1$ do
    Find a minimum-weight edge $e^\star = (v^\star, u^\star)$ among all the edges $(v, u)$ such that $v$ is in $V_T$ and $u$ is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^\star\}$
    $E_T \leftarrow E_T \cup \{e^\star\}$
  end for
  Return $E_T$
```

Algorithm 7 Kruskal(G)

```
Require: A weighted connected graph $G = \langle V, E \rangle$
Ensure: $E_T$, the set of edges composing a minimum spanning tree of $G$
  Sort $E$ in nondecreasing order of the edge weights $w(e_{i_1}, \le \ldots, w(e_{i_{|E|}}))$
  $E_T \leftarrow \emptyset$
  $ecounter \leftarrow 0$      ▷ Initialize the set of tree edges and its size
  $k \leftarrow 0$      ▷ Initialize the number of processed edges
  while $ecounter < |V| - 1$ do
    $k \leftarrow k + 1$
    if $E_T \cup \{e_{i_k}\}$ is acyclic then
      $E_T \leftarrow E_T \cup \{e_{i_k}\}$
      $ecounter \leftarrow ecounter + 1$
    end if
  end while
```

ชื่อ-สกุล ________________________________ รหัสนักศึกษา ____________________

Iterative Improvement

Algorithm 8 MaximumBipartiteMatching(G)

```
  *Finds a maximum matching in bipartite graph by a BFS-like traversal*
Require: A bipartite graph $G = \langle V, U, E \rangle$
Ensure: A maximum-cardinality matching $M$ in the input graph
  Initialize set $M$ of edges with some valid matching (e.g., the empty set)
  Initilize queue $Q$ with all the free vertices in $V$ (in any order)
  while not $Empty(Q)$ do
      $w \leftarrow Front(Q)$
      $Dequeue(Q)$
      if $w \in V$ then
          for every vertex $u$ adjacent to $w$ do
              if $u$ is free then
                  //Augment
                  $M \leftarrow M \cup (w, u)$
                  $v \leftarrow w$
                  while $v$ is labeled do
                      $u \leftarrow$ vertex indicated by $v$'s label
                      $M \leftarrow M - (v, u)$
                      $v \leftarrow$ vertex indicated by $u$'s label
                      $M \leftarrow M \cup (v, u)$
                  end while
                  Remove all vertex labels
                  Reinitialize $Q$ with all free vertices in $V$
                  Break                                          ▷ exit the for loop
              else                                               ▷ u is unmatched
                  if $(w, u) \notin M$ And $u$ is unlabeled then
                      Label $u$ with $w$
                      $Enqueue(q, u)$
                  end if
              end if
          end for
      else                                                       ▷ $w \in U$ (and matched)
          Label the mate $v$ of $w$ with $w$
          $Enqueue(Q, v)$
      end if
  end while
  Return $M$                                                     ▷ current matching is maximum
```

Algorithm 9 Stable Marriage Algorithm

```
Require: A set of $n$ men and a set of $n$ women along with rankings of the women by each man
  and rankings of the men by each woman with no ties allowed in the rankings
Ensure: A stable marriage matching
  Start with all the men and women being free
  while there are free men, arbitrary select one of them and do
      Proposal: The selected free man $m$ proposes to $w$, the next woman on his preference list (who
  is the highest-ranked woman who has not rejected him before).
      Response: If $w$ is free, she accepts the proposal to be matched with $m$. If she is not free, she
  compares $m$ with her current mate. If she prefers $m$ to him, she accepts $m$'s proposal, making
  her former mate free; otherwise, she simply rejects $m$'s proposal, leaving $m$ free.
  end while
  Return the set of $n$ matched pairs.
```