1. *Problem 1: Complementary slackness*

Consider the following linear program in canonical form

$$\begin{array}{lrcrcrcrcrcrcl}
\max & & & 4x_2 & + & 2x_3 & - & x_4 & + & 3x_5 & - & 7x_6 & & \\
\text{s.t.} & x_1 & & & & & - & x_4 & & & + & 3x_6 & \leq & -1 \\
& 7x_1 & + & x_2 & + & x_3 & & & + & x_5 & + & x_6 & \leq & 5 \\
& & & x_2 & - & x_3 & & & & & - & x_6 & \leq & 1 \\
& & & x_2 & - & x_3 & + & x_4 & + & x_5 & + & x_6 & \leq & 5 \\
& x_1 & & & + & x_3 & - & 4x_4 & + & 5x_5 & & & \leq & 0 \\
& & & & & & & & & & & x_i & \geq & 0 \quad \forall i \in \{1, \dots, 6\}
\end{array} \tag{1}$$

1.a *Write the dual of the above linear program.*

$$\begin{array}{lrcrcrcrcrcrcl}
\min & - & y_1 & + & 5y_2 & + & y_3 & + & 5y_4 & & & & \\
\text{s.t.} & & y_1 & + & 7y_2 & & & & & + & y_5 & \geq & 0 \\
& & & & y_2 & + & y_3 & + & y_4 & & & \geq & 4 \\
& & & & y_2 & - & y_3 & - & y_4 & + & y_5 & \geq & 2 \\
& - & y_1 & & & & & + & y_4 & - & 4y_5 & \geq & -1 \\
& & & & y_2 & & & + & y_4 & + & 5y_5 & \geq & 3 \\
& & 3y_1 & + & y_2 & - & y_3 & + & y_4 & & & \geq & -7 \\
& & & & & & & & & & y_i & \geq & 0 \quad \forall i \in \{1, \dots, 5\}
\end{array} \tag{2}$$

1.b *Consider the primal feasible solutions*

$$x^* = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 4 \\ 0 \\ 1 \end{pmatrix} \qquad \text{and} \qquad x^* = \begin{pmatrix} 0 \\ 3 \\ 2 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Use the complementary slackness theorem (Theorem 1.90 in the script) to decide whether these two candidates are optimal solutions of the given linear program.

1.b.1 First x^*

The objective value for the primal solution x^* is $4 \cdot 2 + 2 \cdot 2 - 4 - 7 = 1$. We first need to derive the dual solution. $y^* = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 14 \end{pmatrix}$ which yields the objective value 2. $2 \neq 1$ and therefore is x^* not optimal.

1.b.2 Second x^*

The objective value for the primal solution x^* is $3 \cdot 4 + 2 \cdot 2 - 1 = 15$ We first need to derive the dual solution $y^* = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 3 \\ 2 \end{pmatrix}$ yields the objective value 15. $15 = 15$ and therefore is x^* is optimal.

2. *Problem 2: Asymptotic growth and Landau notation*

Indicate whether the following statements are true or false by ticking the right box. Additionally, provide a short justification of your answer (the given space should be enough).

2.a *Let $f(n) = \log(n^2)$ and $g(n) = \log(n)$. Then $f = \Theta(g)$*

[✓ True] ☐ False

Short justification: Since $\log(n^2) = 2\log(n)$, $f(n)$ grows faster by a constant factor of 2.

2.b *Let $f(n) = n + \sqrt{n}$ and $g(n) = n\log(n)$. Then $f = O(g)$.*

[✓ True] ☐ False

Short justification: $n + \sqrt{n} < n\log(n)$ and therefore $f(n) < g(n)$.

2.c *Let $f(n) = 2^n$ and $g(n) = n!$. Then $f = \Theta(g)$.*

☐ True [✓ False]

Short justification: $n!$ grows faster than 2^n

2.d *Let $f(n) = \sqrt{n}^n$ and $g(n) = n^{\sqrt{n}}$. Then $f = \Theta(g)$.*

☐ True [✓ False]

Short justification: $\sqrt{n}^n = n^{\frac{n}{2}}$ and $n^{\sqrt{n}} = n^{n^{\frac{1}{2}}}$. Clearly $\frac{n}{2}$ does not grow the same as $n^{\frac{1}{2}}$.

2.e *Let $f(n) = 3^n$ and $g(n) = 2^{n+\log n}$. Then $f = \Omega(g)$*

. [✓ True] ☐ False

Short justification: 3^n grows faster than $2^{n+\log n}$

2.f *Let $f, g : \mathbb{Z}_{\geq 1} \to \mathbb{Z}_{\geq 1}$. Then, at least one of the relations $f = O(g), g = O(f)$, or $f = \Theta(g)$ holds.*

[✓ True] ☐ False

Short justification: If f grows faster than g, $f = O(g)$ holds. If g grows faster than f, $g = O(f)$ holds. If both grow equally fast $f = \Theta(g)$ holds.

3. *Problem 3: Running time of a sorting algorithm*

. . . insertion sort . . .

Implementation (simplified for myself)

```
InsertionSort(int[] A) {
   for (int i = 1; i < A.length; ++i) {
      int j = i;
      while (j ≥ 1 && A[j-1] > A[j]) {
         int x = A[j];
         A[j] = A[j-1];
         A[j-1] = x;
         j--;
      }
   }
   return A;
}
```

3.a *Track how a call of Algorithm 1 on the array $A = [4, 3, 1, 2]$ changes A*

$[4, 3, 1, 2]$
$[3, 4, 1, 2]$
$[3, 1, 4, 2]$ $[1, 3, 4, 2]$
$[1, 3, 2, 4]$ $[1, 2, 3, 4]$

3.b *Prove that Algorithm 1 has a worst-case running time of $O(n^2)$.*

Worst case scenario is with a reverse-sorted array. In this case it performs i steps i times. Therefore the running time is $i \cdot i = i^2 = O(n^2)$

3.c *Show that Algorithm 1 has a running time of $\Omega(n^2)$ on some inputs*

$A = [4, 3, 2, 1]$

3.d *Show that there exist inputs on which Algorithm 1 has a running time of only $O(n)$.*

Sorted array $A = [1, 2, 3, 4]$

4. *Problem 4: Quering edge existence in incidence lists*

Let $G = (V, E)$ be a graph given by an incidence list and let $u, v \in V$ two vertices. Show that in time $O(\min\{deg(u), deg(v)\})$, it can be decided whether G contains an edge $\{u, v\}$.

The edge $\{u, v\}$ is contained in the list for u and in the list for v. An algorithm with running time of $O(2 \cdot \min\{deg(u), deg(v)\})$ woks as the following:

```
ContainsEdge(u, v) {
   var currentUEdge = u.FirstEdge;
   var currentVEdge = v.FirstEdge;
   while(true) {
      if (currentUEdge.Start == v || currentUEdge.End == v) {
         return true;
         if (!currentUEdge.Next()) {
            return false;
         }
         currentUEdge = currentUEdge.Next();
      }
      if (currentVEdge.Start == u || currentVEdge.End == u) {
         return true;
         if (!currentVEdge.Next()) {
            return false;
         }
         currentVEdge = currentVEdge.Next();
      }
   }
}
```

5. *Problem 5: Meeting at a central point*

Imagine three caterpillars sitting on vertices $v1, v2$, and $v3$ of a connected graph $G = (V, E)$. The three caterpillars would like to meet at one of the vertices of the graph. All they can do is moving from one vertex to another via the edges of the graph, and they can only stop at vertices of the graph,but not on an edge. When traversing an edge, the three caterpillars travel at the same speed of one edge per hour.

For an example, consider Figure 1. To meet at vertex $v5$, the caterpillar from vertex $v1$ could travel to $v5$ via the vertices $v4$ and $v2$ in three hours (and there is no faster option), and the caterpillars from $v2$ and $v3$ could be there in one and two hours, respectively. Thus, an earliest meeting at $v5$ could take place three hours after the caterpillars start moving.

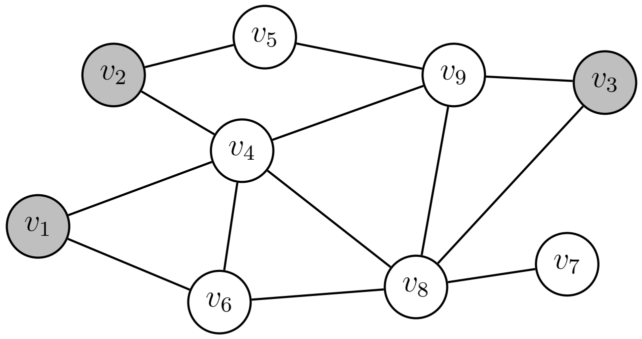

Figure 1: A graph $G = (V, E)$ with starting points of the caterpillars marked in gray.

Of course, the caterpillars want to meet at a vertex where an earliest possible meeting can be achieved. Can you provide an algorithm that determines such a vertex? Analyze the running time of your algorithm. Is it possible to construct an algorithm that solves the given problem and has running time $O(|V| + |E|)$?

Algorithm idea in words: It's a BFS algorithm, we cycle through the caterpillars last visited Verteces and move 1 further. If we get to an edge where all 3 caterpillars join, we have found the minimum point. To simplify the technical details, the class *Vertex* is given as the following:

```
class Vertex {
   public int?[] Distance = new int?[caterpillars.Count];
   public Vertex[] adjacentVerteces();
}
```

and *caterpillars* is an array of the class *Caterpillar* given as the following:

```
class Caterpillar {
   public Vertex StartingVertex;
}
```

Algorithm:

```
var latestCaterpillarVertices = new Vertex[caterpillars.Count][]
for (var i = 0; i < caterpillars.Count; ++i) {
   latestCaterpillarVertices[i] = new [] {caterpillars[i].
      StartingVertex};
}

currDistance = 0;
while (true) {
   currDistance++;
   for (var i = 0; i < caterpillars.Count; ++i) {
      var newLatestCaterpillarVertices = [];

      foreach (var vertex in latestCaterpillarVertices[i]) {
         foreach (var adjacentVertex in vertex.adjacentVerteces()) {
            if (null == adjacentVertex.Distances[i]) {
               // vertex not yet visited
               adjacentVertex.Distances[i] = currDistance;
               newLatestCaterpillarVertices.Add(adjacentVertex);

               // check break condition
               meetingPointFound = true;
               for (var j = 0; j < caterpillars.Count; ++j) {
                  meetingPointFound &= null ≠ adjacentVertex.
                     Distances[i];
               }
               if (meetingPointFound) {
                  return adjacentVertex;
               }
            }
         }
      }
      latestCaterpillarVertices[i] = newLatestCaterpillarVertices;
   }
}
```

The running time is $O(\#caterpillars \cdot (|V| + |E|))$