Aufgabe 1 [2+2+2 Punkte] Welche der folgenden Aussagen gelten? Begründen Sie Ihre Aussage jeweils kurz.

(a) Jeder zusammenhängende ungerichtete Graph, der weniger Kanten als Knoten hat, ist bipartit.

(b) Für alle positiven Zahlen $x, y \in F_{\texttt{double}}$ ist auch $x - y \in F_{\texttt{double}}$.

(c) Sei $n \in \mathbb{N}$ und $f : \{0, 1, \ldots, 2^n\} \to \{0, 1\}$ mit $f(0) = 0$ und $f(2^n) = 1$. Dann kann man durch Abfrage von n Funktionswerten eine Zahl $x \in \{0, \ldots, 2^n - 1\}$ finden mit $f(x) < f(x + 1)$.

Aufgabe 2 [3+3 Punkte]

(a) Geben Sie eine ungerade Zahl a an, so dass die Berechnung von $\mathrm{ggT}(5, a)$ mit dem Euklidischen Algorithmus möglichst viele Iterationen braucht. Listen Sie die Zwischenergebnisse auf.

(b) Geben Sie eine Folge von acht verschiedenen natürlichen Zahlen an, für deren Sortierung Merge-Sort möglichst viele Vergleiche benötigt. Zeigen Sie, wie Merge-Sort diese Zahlen sortiert.

Eine Begründung ist jeweils nicht erforderlich.

Aufgabe 3 [2+2+2 Punkte] Geben Sie (ohne Beweis) polynomielle Algorithmen für folgende Entscheidungsprobleme an. Gegeben ist jeweils ein Digraph G; sei $n := |V(G)|$. Sie dürfen Algorithmen aus der Vorlesung verwenden.

(a) Enthält G ein Branching mit genau $n - 2$ Kanten?

(b) Enthält G einen (gerichteten) Kreis mit weniger als $\frac{n}{2}$ Kanten?

(c) Ist $\sum_{v \in V(G)} (|\delta^+(v)| + |\delta^-(v)|)$ eine Primzahl?

Aufgabe 4 [6 Punkte] Sei G ein gerichteter Graph, $F \subseteq E(G)$, und $s, t \in V(G)$. Für jedes $k \in \mathbb{N}$ betrachten wir das Netzwerk $N_k := (G, u_k, s, t)$, wobei $u_k(e) := k$ für $e \in F$ und $u_k(e) := 1$ für $e \in E(G) \setminus F$ sei. Sei V_k der maximale Wert eines s-t-Flusses in N_k. Beweisen Sie: $\lim_{k \to \infty} V_k = \infty$ genau dann, wenn t von s aus in $(V(G), F)$ erreichbar ist.

Aufgabe 5 [6 Punkte] Sei $A = (\alpha_{ij}) \in \mathbb{R}^{n \times n}$ eine Matrix mit $2|\alpha_{ii}| > \sum_{j=1}^{n} |\alpha_{ij}|$ für alle $i = 1, \ldots, n$. Beweisen Sie, dass A invertierbar ist und eine LU-Zerlegung besitzt.

Aufgabe 6 [10 Punkte] Ergänzen Sie das folgende Programmstück in C++ (Zeilen 10–34) so, dass es korrekt testet, ob der gegebene gerichtete Graph eine Arboreszenz ist, deren Wurzel der Knoten mit Index 0 ist. Außer den aus der Vorlesung bekannten Klassen `Graph` (siehe anliegenden Ausdruck) und `vector` dürfen Sie nichts verwenden.

```
1  #include "graph.h"
2
3
4  bool is_arborescence_with_root_zero(Graph const & g)
5  {
6       if ((g.num_nodes() < 1) or (g.get_node(0).in_edges().size() > 0))
7       {
8            return false;
9       }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35 }
36
37
38 int main(int argc, char* argv[])
39 {
40      if (argc > 1)
41      {
42           Graph digraph = read_graph(argv[1]);
43           print_graph(digraph);
44           if (is_arborescence_with_root_zero(digraph))
45           {
46                std::cout << "Der Digraph ist eine Arboreszenz mit Wurzel 0.\n";
47           }
48           else
49           {
50                std::cout << "Der Digraph ist keine Arboreszenz mit Wurzel 0.\n";
51           }
52      }
53      return 0;
54 }
```

Aufgabe 1

(a): wahr: Ein zusammenhängender Graph mit n Knoten enthält einen aufspannenden Baum, also mindestens $n-1$ Kanten. Enthält er genau $n-1$ Kanten, so ist der Graph selbst ein Baum, also kreisfrei, und somit bipartit.

(b): falsch: z.B. für $x = 2$ und $y = 2^{-53}$ ist $x - y = \sum_{i=0}^{53} 2^{-i}$ nicht in $F_{\texttt{double}}$.

(c): wahr: dies leistet die binäre Suche, die mit $L = 0$ und $U = 2^n$ startet und – solange $U > L + 1$ ist – jeweils L oder U auf $M := \frac{L+U}{2}$ setzt, je nachdem ob $f(M)$ null oder eins ist. Der Beweis folgt unmittelbar per Induktion über n.

Aufgabe 2

(a): dies leisten die Zahlen 13, 23, 33 usw., zum Beispiel ist $\mathrm{ggT}(5, 13) = \mathrm{ggT}(5, 3) = \mathrm{ggT}(2, 3) = \mathrm{ggT}(2, 1) = 1$.

(b): die Folge 1,5,3,7,2,6,4,8 wird in 1,5,3,7 und 2,6,4,8 zerlegt, diese in 1,5 und 3,7 und 2,6 und 4,8. Man braucht je einen Vergleich um zu garantieren, dass diese Zweierfolgen sortiert sind, je drei für die beiden Merge-Schritte (1-3, 3-5, 5-7 bzw. 2-4, 4-6, 6-8), die damit die sortierten Viererfolgen 1,3,5,7 und 2,4,6,8 berechnen, und schließlich sieben Vergleiche (je zwei aufeinanderfolgende Zahlen) für deren Verschmelzen. Insgesamt also $4 + 2 \cdot 3 + 7 = 17$ Vergleiche.

Aufgabe 3

(a): Ergänze eine neue Kante (u, v), wobei u und v zwei Knoten von G seien (für u und v werden alle $\binom{n}{2}$ Möglichkeiten ausprobiert), und prüfe (mit Graphendurchmusterung), ob es eine Arboreszenz mit Wurzel u gibt.

(b): Durch BFS von jedem Knoten aus kennt man alle Abstände. Nun prüft man, ob eine Kante (v, w) existiert mit $\mathrm{dist}(w, v) < \frac{n}{2} - 1$.

(c): Da $\sum_{v \in V(G)} (|\delta^+(v)| + |\delta^-(v)|) = 2|E(G)|$ stets gerade ist, ist sie genau dann eine Primzahl, wenn der Graph nur eine Kante enthält.

Aufgabe 4

Gibt es einen s-t-Weg P in $(V(G), F)$, so kann man $f(e) = k$ für $e \in E(P)$ und $f(e) = 0$ für $e \in E(G) \setminus E(P)$ setzen und erhält einen s-t-Fluss in N_k vom Wert k, also $V_k \geq k$ für alle $k \in \mathbb{N}$.

Enthält umgekehrt die von s aus in $(V(G), F)$ erreichbare Knotenmenge R nicht t, so ist der Wert eines s-t-Flusses in N_k für alle $k \in \mathbb{N}$ durch $u_k(\delta^+(R)) = |\delta^+(R)|$ beschränkt.

Aufgabe 5

Die Gauß-Elimination braucht hier keine Spalten- oder Zeilenvertauschungen, denn die genannte Bedingung bleibt zu jedem Zeitpunkt erhalten: durch Subtraktion des

$\frac{\alpha_{ir}}{\alpha_{rr}}$-fachen der r-ten Zeile von der i-ten Zeile wird $|\alpha_{ii}| - \sum_{j \notin \{i,r\}} |\alpha_{ij}|$ höchstens so stark verringert wie $|\alpha_{ir}|$. Die Diagonalelemente bleiben somit von null verschieden, und am Ende erhält man eine LU-Zerlegung, wobei neben L auch U nichtsingulär ist.

Aufgabe 6

```cpp
#include "graph.h"


bool is_arborescence_with_root_zero(Graph const & g)
{
     if ((g.num_nodes() < 1) or (g.get_node(0).in_edges().size() > 0))
     {
          return false;
     }

     std::vector<Graph::NodeId> reached;
     reached.push_back(0);
     size_t count = 0;

     while (count < reached.size())
     {
          Graph::Node current_node = g.get_node(reached[count]);

          for (Graph::EdgeId i = 0;
               i < g.get_node(reached[count]).out_edges().size(); ++i)
          {
               Graph::NodeId new_nodeid =
                    g.get_edge(g.get_node(reached[count]).out_edges()[i]).get_head();
               reached.push_back(new_nodeid);
               if (g.get_node(new_nodeid).in_edges().size() > 1)
               {
                    return false;
               }
          }

          count++;
     }

     return (count == g.num_nodes());
}


int main(int argc, char* argv[])
{
     if (argc > 1)
     {
          Graph digraph = read_graph(argv[1]);
          print_graph(digraph);
          if (is_arborescence_with_root_zero(digraph))
          {
               std::cout << "Der Digraph ist eine Arboreszenz mit Wurzel 0.\n";
          }
          else
          {
               std::cout << "Der Digraph ist keine Arboreszenz mit Wurzel 0.\n";
          }
     }
     return 0;
}
```