Die Lösungsidee:

Um die minimale LLL für eine bestimmte Biberverteilung zu bestimmen, ist es am einfachsten, einfach alle möglichen Telepaartien, die gemacht werden können, auszuprobieren, bis eine Telepaartie stattfindet, durch die ein Behälter leer wird. Man probiert also alle möglichen Telepaartien für die Startverteilung aus und für jede dadurch entstehenden Verteilungen probiert man wieder alle möglichen Telepaartien aus. Wie sich sehen lässt, baut diese Idee auf Rekursion. Es wird immer wieder die selbe Methode mit unterschiedlichen Verteilungen aufgerufen, bis es möglich wird, einen Behälter gänzlich zu leeren.
Diese Methode funktioniert jedoch auf den bisher beschriebenen Weg noch nicht: es könnte schließlich zu einer unendlichen Schleife kommen, bei welcher immer wieder die selben Verteilungen vorkommen und es nie zu einer Lösung kommt. Um dieses Problem zu umgehen, kann man die Rekursion nach einer bestimmten Anzahl rekursiver Aufrufe beenden. Diese Anzahl muss zunächst von außen festgelegt werden. Im Laufe der Bearbeitung der Aufgabe konnte ich feststellen, dass es keine LLL mit Wert größer 10 zu geben scheint. Da ich hierfür aber keinen konkreten Beweis habe, ist mit dieser Zahl vorsichtig umzugehen. In der Implementation habe ich deswegen die Möglichkeit gegeben, eine Telepaartie Obergrenze am Anfang des Programms einzugeben. Für die Testfälle habe ich eine Obergrenze von 25 gewählt.

Die Obergrenze verändert sich anschließend zu der kleinsten bereits gefundenen LLL. In anderen Worten: jedes Mal, wenn eine Telepaartie durchgeführt wird, die einen Behälter leert, welche nach weniger Telepaartien zu diesem Ergebnis kam, als die Obergrenze zuließ, wird die Obergrenze auf die Anzahl der zuletzt benötigten Telepaartien gesetzt. Dadurch bricht der Algorithmus rekursive Aufrufe der Methode immer schneller werdend ab, wenn sie nicht mehr zu einer optimaleren Lösung werden können.

Um nun aus allen LLLn für eine bestimmte Gesamtanzahl an Bibern die größte LLL zu finden, muss lediglich für jede Startverteilung, die mit der gegebenen Biberanzahl möglich ist, die entsprechende minimale LLL bestimmt werden. Aus all diesen Ergebnissen muss dann der größte Wert ausgegeben werden.

An dieser Stelle muss ich gestehen, dass mein Algorithmus für den zweiten Teil nicht gänzlich optimal ist, da er zum Beispiel die Startverteilungen [1,4,5] wie auch [4,1,5] testet, obwohl beide letztlich zum identischen Ergebnis kommen, da sie letztlich nur unterschiedlich sortiert sind. Aus zeitlichen Gründen war es mir aber nicht möglich dies noch zu optimieren. Für die gegebene Biberanzahl n=100 im Testfall genügt die jetzige Implementation dennoch, um in angemessener Zeit zu einem Ergebnis zu gelangen (unter 2 Sekunden auf meinem Computer).

Die Umsetzung mit Quelltext:

Die Umsetzung der Lösungsidee erfolgt in meiner Bearbeitung in der Programmiersprache Ruby. Das Programm liegt im Ordner „Telepaartie“ und stellt zeitgleich Quelltext und ausführbare Datei dar. Ich habe das Programm unter macOS 10.14.6 und der Ruby Version 2.3.7p456 (2018-03-28 revision 63024) [universal.x86_64-darwin18] getestet.

In den ersten zwei Zeilen des Programms wird die oben beschriebene Obergrenze an Telepaartien abgefragt. Dieser Wert wird in „$min_saved“ zwischengespeichert.

In Zeile drei wird die Variable „$min“ für den späteren Gebrauch in der darauf folgenden Methode „LLL“ von Zeile 4 bis 34 definiert. In den Zeilen 36 bis 53 befindet sich des Weiteren die Methode „L“. Alle darauf folgenden Zeilen dienen ausschließlich der interaktiven Ein- und Ausgabe des Programms. Aus diesem Grund werde ich nur auf die Zeilen 4 bis 53 im Folgenden genauer eingehen.

Die Methode „LLL“ bekommt vier Werte übergeben: die drei Anzahlen an Bibern in den verschiedenen Behältern („l1“, „l2“ und „l3“) sowie die Information, wie viele Telepaartien bereits verwendet wurden, um diese Biberverteilung zu erreichen („i“).

```
def LLL(l1, l2, l3, i)
  if(l1 == 0 or l2 == 0 or l3 == 0)
    $min = i
```

In jedem Aufruf der Methode wird dann zunächst kontrolliert, ob einer der Behälter leer ist. Ist dies der Fall wird „$min“ mit der Anzahl an bis zu diesem Punkt benötigten Telepaartien überschrieben. „$min“ steht dabei entweder für die kleinste Anzahl an benötigten Telepaartien, um einen Behälter zu leeren, welche bisher gefunden wurde oder die vorher festgelegte Obergrenze, wenn noch keine zielführende Lösung gefunden wurde.

```
  else
    if(i < $min)
      if l1 == l2
        LLL(l1+l2, 0, l3, i+1)
      elsif l2 == l3
        LLL(l1, l2+l3, 0, i+1)
      elsif l3 == l1
        LLL(0, l2, l3+l1, i+1)
```

Sollte hingegen kein Behälter leer sein, so wird überprüft, ob zwei der Behälter die selbe Anzahl an Bibern haben. Ist dies nämlich der Fall lassen sich alle Biber des einen Behälters in den anderen telepaartieren. Dementsprechend findet an dieser Stelle dann ein rekursiver Aufruf statt, bei dem abhängig davon, welche Behälter die selbe Anzahl an Bibern haben, die Biber des einen davon in den anderen telepaartiert werden.

Hierbei ist anzumerken, dass dieser gesamte Abschnitt nur weitere rekursive Aufrufe startet, wenn die derzeitige Anzahl an Telepaartien „$min“ nicht übersteigt. Es handelt sich also um eine Abbruchbedingung für auftretende Schleifen beim Telepaartieren und beschleunigt den Algorithmus zum Ende hin, da „$min“ durch das finden besserer LLL immer kleiner wird, wodurch weniger rekursive Aufrufe zugelassen werden.

```
        else
            if l1 > l2
                LLL(l1-l2, l2*2, l3, i+1)
            else
                LLL(l1*2, l2-l1, l3, i+1)
            end
            if l2 > l3
                LLL(l1, l2-l3, l3*2, i+1)
            else
                LLL(l1, l2*2, l3-l2, i+1)
            end
            if l3 > l1
                LLL(l1*2, l2, l3-l1, i+1)
            else
                LLL(l1-l3, l2, l3*2, i+1)
            end
        end
    end
  end
end
```

Sollte keiner der vorherigen Zustände in der übergebenen Verteilung vorliegen, so werden alle drei möglichen Telepaartien für die derzeitige Verteilung rekursiv aufgerufen/ausprobiert. Dabei wird stets kontrolliert, welche Behälter mehr Biber enthält, da dies entscheidet, von welchen Behälter in welchen anderen Behälter die Biber telepaartiert werden.

Die Methode „L“ erhält die Gesamtbiberzahl „n“, mit welcher alle, mit dieser Anzahl möglichen, Startverteilungen auf ihre LLL überprüft werden.

```
def L(n)
  max = 0
```

Da die größte LLL für eine Gesamtbiberanzahl gesucht wird, wird zunächst eine Variable mit dem Wert *0* definiert. Diese wird später die größte gefundene LLL enthalten.

```
  n1 = 1
  while(n1 < n)
    n2 = 1
    while(n2 < n-n1)
      $min = $min_saved
      LLL(n1, n2, n-n1-n2, 0)
      puts "LLL für (#{n1}, #{n2}, #{n-n1-n2}): #{$min}"
      if $min > max
        max = $min
      end
      n2 += 1
    end
    n1 += 1
  end
  return max
end
```

Abschließend folgen zwei verschachtelte Schleifen, welche dazu dienen alle Startverteilungen einmal aufzurufen. Dabei wird jedes mal vor dem Aufruf der Methode „LLL“ „$min“ auf „$min_saved“ gesetzt, um die Obergrenze an Telepaartien wieder festzulegen. Am Ende eines jeden Durchlaufs der Rekursion der Methode „LLL“ ist in „$min“ letztlich die benötigte LLL für die übergebene Startverteilung gespeichert. Ist diese größer als „max“ wird „max“ mit „$min“ überschrieben. So kann die maximale LLL für die Biberanzahl „n“ zuletzt zurückgegeben werden.

Beispiel/Demonstration:

Im folgenden möchte ich zusammengefasst zeigen, was in der .txt-Datei im Ordner des Programms, welche die genauen Ausgaben des Programms enthält, geschrieben steht. Die gesamte Ausgabe ist dadurch, dass bei dem Aufruf der Methode „L“ Zwischenergebnisse der verschiedenen gefundenen LLLn ausgegeben werden deutlich länger und ist deshalb hier nicht einfach hinein kopiert.

LLL(2, 4, 7) = 2
LLL(3, 5, 7) = 3
LLL(80, 64,32) = 2

L(10) = 2
L(100) = 7

Wie bereits gesagt bitte ich darum, dass für die genaue Ausgabe die .txt-Datei im Ordner des Programms wahrgenommen wird. Natürlich lassen sich die selben Ergebnisse aber auch durch das Ausführen des Programms erreichen.