D.1 Untersuchen in den folgenden Teilaufgaben das Problem MINIMUM INTERVAL PARTITIONING. Die Eingabe ist identisch zu Problem MAXIMUM INTERVAL SCHEDULING aus der Vorlesung, jedoch müssen jetzt *alle* m Intervalle berücksichtigt werden. Dafür stehen bis zu m identische Ressourcen zur Verfügung. Die einer Ressource zugeordneten Intervalle müssen wie gehabt überschneidungsfrei sein.

4. Implementieren Sie einen Algorithmus `min_intpart_greedy` nach dem *Greedy*-Entwurfsmuster, der die Intervalle nach Startzeiten aufsteigend sortiert betrachtet. Sie können davon ausgehen, dass die Eingabe bereits sortiert vorliegt. Die Greedy-Auswahlregel lautet: Wähle die kleinste Ressourcennummer, die (wieder) frei ist. Liefern Sie nur den Wert der Greedy-Lösung zurück und geben Sie die Lösung selbst mit **`print`** aus. Analysieren Sie die Laufzeit. Wenn Sie nicht $O(m^2)$ erreichen, verbessern Sie Ihren Algorithmus.

```
5   # Enwurfsmuster Greedy
6   # Laufzeit: O(m^2)
7   def min_intpart_greedy(L):
8       m = len(L)                                  # O(1)
9       R = [{0}]                                   # O(1)
10      (_,last) = L[0]                             # O(1)
11      Lasts = [last]                              # O(1)
12
13      for i in range(1,m):                        # O(m)
14          (s,f) = L[i]                                # O(1)
15          ressource_found = False                     # O(1)
16          for j in range(len(Lasts)):                 # O(m)
17              if s >= Lasts[j]:                           # O(1)
18                  R[j].add(i)
19                  Lasts[j] = f
20                  ressource_found = True
21                  break
22
23          if not ressource_found:                     # O(1)
24              R.append({i})
25              Lasts.append(f)
26
27
28      return R
```

5. Argumentieren Sie, weshalb Ihr Greedy-Algorithmus nur *zulässige* Lösungen konstruiert:

   (a) Wird jedem Intervall *höchstens* eine Ressource zugeordnet?

Ja, denn die Liste der Intervalle L wird nur einmal durchlaufen (Zeile 13). Anschließend wird nach einem freien Platz in der Liste der Ressourcen gesucht (Zeile 17). Nur wenn kein freier Platz gefunden wird, wird eine neue Ressource erstellt und das Intervall wird der neuerstellten Ressource hinzugefügt (Zeile 23)

→ Somit wird jedes Intervall höchstens einer Ressource zugeordnet

   (b) Wird jedem Intervall *mindestens* eine Ressource zugeordnet?

Ja, denn die Liste der Intervalle L wird in Zeile 13 komplett durchlaufen. Anschließend wird nach einem freien Platz in der Liste der Ressourcen gesucht (Zeile 17). Wird kein freier Platz gefunden, wird eine neue Ressource erstellt und das Intervall wird der neuerstellten Ressource hinzugefügt (Zeile 23)

→ Somit wird jedes Intervall mindestens einer Ressource zugeordnet

   (c) Haben zwei überlappende Intervalle verschiedene Ressourcennummern?

Ja, denn für jede Ressource wird in der Liste $Lasts$, die Endzeit f des spätesten Intervalls festgehalten, sodass nur diejenigen Intervalle zur Ressource hinzugefügt werden, deren Startzeit kleiner als die in Lasts festgehaltene Endzeit ist (Zeile 17).

→ Zwei überlappende Intervalle haben verschiedene Ressourcennummern

6. Sei d die maximale Anzahl von Intervallen in der Eingabe, die sich paarweise überlappen. Beweisen Sie mit Hilfe der folgenden Teilaufgaben, dass Ihr Greedy-Algorithmus eine *optimale* Lösung konstruiert:

(a) Jede zulässige Lösung benötigt mindestens d Ressourcen, also auch jede optimale Lösung.

Wie in Aufgabenteil 5.c argumentiert, haben zwei überlappende Intervalle verschiedene Ressourcennummern. Wenn d die maximale Anzahl von Intervallen in der Eingabe ist, die sich paarweise überlappen, dann müssen all diese Intervalle unterschiedliche Ressourcennummern haben.

Es existieren also d Intervalle mit unterschiedlichen Ressourcennummern und somit mindestens d Ressourcen.

(b) Durch Widerspruchsbeweis: Der Greedy-Algorithmus verwendet höchstens die Ressourcen $0, \ldots, d-1$.

Angenommen der Greedy-Algorithmus verwendet die Ressourcen $0, \ldots, d$:

Die maximale Anzahl von Intervallen in der Eingabe, die sich paarweise überlappen ist d. Seien $I_d = \{i_0, \ldots, i_{d-1}\}$ diese d-Intervalle. Nun wird jedes dieser Intervalle einer unterschiedlichen Ressource zugeordnet:

$i_0 \rightarrow R_0$

$i_1 \rightarrow R_1$

…

$i_{d-1} \rightarrow R_{d-1}$

Die Ressource R_d enthält keines dieser Intervalle. Weil kein weiteres Intervall in der Eingabe existiert, das die Intervalle I_D paarweise überlappt, ist die Ressource R_d immer leer, da zu jedem Zeitpunkt eine frühere Ressource frei ist.

Somit ist die Ressource R_d überflüssig, was zu einem Widerspruch der Annahme führt.

q.e.d.