Storyteller Subtask 4

N gerade:

mouses_amount ist die Anzahl Mäuse, also N und im Array mouses sind die Erzählzeiten aus dem Input gespeichert. In mouses_time werden die Startpunkte des Erzählens gespeichert.

```
for j in range(mouses_amount):
  max_time = max(max_time, mouses[j-1]+mouses[j])
  if j%2==0:
      mouses_time.append(0)
  else:
      if j+1 == mouses_amount:
          mouses_time.append(max(mouses[j-1],mouses[0]))
      else:
          mouses_time.append(max(mouses[j-1],mouses[j+1]))
amount = max_time
```

Der Algorithmus berechnet die kürzeste Zeit T, damit alle Mäuse ihre Geschichte erzählen können, indem er immer die Zeiten zweier Nachbarn addiert und dann schaut, ob diese Zeit grösser ist, als die bisherige längste kürzeste Zeit T. Wenn dies zutrifft, wird die neue Zeit der Nachbarn zu T, sonst nicht.
Ebenfalls wenn bei Maus i i%2==0 ist, dann fängt diese Maus bei t=0 an zu erzählen. Wenn bei Maus i i%2!=0 ist, dann beginnen die beiden Nachbarmäuse bei t=0 an zu erzählen. Deswegen muss geschaut werden, welcher Nachbar länger braucht und diese Zeit ist dann der Erzählbeginn.

Der Algorithmus hat eine asymptotische Laufzeit von O(n) und einen asymptotischen Speicherverbrauch von O(n) weil einmal über alle Mäuse iteriert werden muss, um T und die Erzählstartpunkte zu berechnen und in diese werden in mouses_time gespeichert, also O(n) Speicher.
Gegeben eine beliebige Eingabe $t_0, \ldots, t_{N-1}$, der Algorithmus ist sowohl:

Obere Schranke für die optimale Lösung: Die vom Algorithmus berechneten Zahlen $y_0, \ldots, y_{N-1}$ erfüllen die Einschränkungen immer $y_i + t_i <= X$ zu sein, wobei X ist die optimale Zeit, dass alle Mäuse ihre Geschichte erzählen können.

Induktion von links nach rechts:
Zu jedem i (0 <= i < N) wissen wir die bisherige minimale Zeit X_{i-1}, dass alle Mäuse bis i-1 ihre Geschichte erzählen können. X_i wird durch $\max(X_{i-1}, t_{i-1} + t_i)$ berechnet. Die Zeit y_i ist für i%2==0 0 und sonst $\max(y_{i-1} + t_{i-1}, y_{i+1} + t_{i+1})$ Die Zeit $y_i + t_i$ kann grösser als X_i sein, aber im nächsten i+1 wird es gleich X_{i+1} sein.

Daher eine optimale Lösung benötigt höchstens X Sekunden.

Untere Schranke für die optimale Lösung: Es ist unmöglich dass eine Lösung weniger als X Sekunden benötigt.

Sei X die Anzahl Sekunden der optimalen Lösung. Wenn $t_{i-1} + t_i > X$ ist, dann ist X nicht die optimale Lösung und somit Falsch. Das ist ein Widerspruch.

Die Lösung ist somit optimal.

N ungerade:

```
min_waiter = [10**10,()]
for j in range(mouses_amount):
  if min_waiter[0] > mouses[j-1] + mouses[j] + mouses[j+1-mouses_amount]:
    min_waiter[0] = mouses[j-1] + mouses[j] + mouses[j+1-mouses_amount]
    if j - 1 == -1:
      min_waiter[1] = (mouses_amount-1,j)
    else:
      min_waiter[1] = (j-1,j)
  max_time = max(max_time, mouses[j-1]+mouses[j])
amount = 10**10
if min_waiter[0] > 0:
  if mouses[min_waiter[1][0]-1] < mouses[min_waiter[1][1]+1-mouses_amount]:
    amount = mouses[min_waiter[1][0]-1] + mouses[min_waiter[1][0]] +
mouses[min_waiter[1][1]]
  else:
    amount = mouses[min_waiter[1][1]+1-mouses_amount] \
+mouses[min_waiter[1][0]] + mouses[min_waiter[1][1]]
  amount = max(max_time, amount)
else:
  amount = max_time

time_first = mouses[min_waiter[1][0]]
time_sec = mouses[min_waiter[1][1]]
time_first_n = mouses[min_waiter[1][0]-1]
time_sec_n = mouses[min_waiter[1][1]+1-mouses_amount]

if min_waiter[1][1] + 1 == mouses_amount:
  j_start = 0
  j = 1
elif min_waiter[1][1] + 2 == mouses_amount:
  j_start = min_waiter[1][1] + 1
  j = 0
```

```
else:
  j_start = min_waiter[1][1] + 1
  j = j_start+1

mouses_time = [0 for x in range(mouses_amount)]

no_zero = True
last_no_zero = mouses[j_start]
while True:
  if j == mouses_amount:
      j = 0
  if no_zero:
      mouses_time[j] = max(last_no_zero, mouses[j+1-mouses_amount])
      no_zero = False
  else:
      last_no_zero = mouses[j]
      mouses_time[j] = 0
      no_zero = True
  if j == j_start:
      #mouses_time[j] = 0
      break
  j+=1
if time_first_n < time_sec_n:
  mouses_time[min_waiter[1][0]] = time_first_n
  mouses_time[min_waiter[1][1]] = max(time_first_n + time_first, time_sec_n)
else:
  mouses_time[min_waiter[1][0]] = max(time_sec_n + time_sec, time_first_n)
  mouses_time[min_waiter[1][1]] = time_sec_n

mouses_time[min_waiter[1][0]-1] = 0
mouses_time[min_waiter[1][1]+1-mouses_amount] = 0
```

Der Algorithmus für N ungerade kann auf N gerade vereinfacht werden, wenn 3 Mäuse entfernt werden. Diese 3 Mäuse wollen natürlich auch ihre Geschichte erzählen und die anderen der Nachbarn hören. Darum wird zu einem bestimmten Zeitpunkt, zwei Mäuse nebeneinander nichts erzählen und ihren Nachbarn zuhören. Sobald der erste andere Nachbar dieser beiden Mäuse fertig ist, beginnt die entsprechende Maus zu erzählen. Dann hört die andere Maus zu und wartet bis beide Nachbaren aufgehört hat und beginnt dann zu erzählen. Der Algorithmus sucht ein solche Paar, bei dem am wenigsten Zeit verloren geht, wenn zwei Mäuse nebeneinander nichts erzählen. Dies wird getan indem zuerst 3 nebeneinanderliegende Mäuse mit deren Erzählzeiten zusammengerechnet minimal sind über die ganzen Mäuse. Dann wird geschaut, ob diese drei Mäuse ihre Geschichte in weniger Zeit als die minimale maximale Erzählzeit, berechnet aus der Zeit zweier

nebeneinanderliegenden Mäusen. Nacher wird geschaut, wann welche Maus dieses Zweierpaars Mäuse und deren Nachbaren und alle anderen Mäuse anfangen sollten zu erzählen.

Die asymptotische Laufzeit beträgt O(n) weil zwei Mal über den Input iteriert wird und der Speicher für die Speicherung des Mauspaares und der Zeiten der Mäuse beträgt asymptotisch auch O(n)