```
[[    0.52   100.   ]
 [    0.52     5.   ]
 [    0.68  1000.   ]
 [    0.52  1000.   ]
 [    0.68    10.   ]
 [    0.52     5.   ]
 [    0.52    50.   ]
 [    0.92    10.   ]
 [    0.76   100.   ]
 [    0.52    50.   ]
 [    0.52   100.   ]
 [    0.6     10.   ]
 [    0.6      5.   ]
 [    0.52  1000.   ]
 [    0.52    10.   ]
 [    0.6      5.   ]
 [    0.6      5.   ]]
optimum response time: 20.16
```

Source code

```
import numpy as np
import math
np.set_printoptions(suppress=True)
signal_num = 0
one_bit_trans = 0
total_qi = 0

def calculate_response(signal_num, one_bit_trans, trans_time,
period_time):
    """[summary]

    Args:
        signal_num (int): signal numbers
        one_bit_trans (float): one bit trans time
        trans_time (array): trans time array size depends on signal
        period_time (array): message period time size => signal num

    Returns:
        worst response (float)
    """
    for i in range(signal_num):
        block_time = np.max(trans_time[i:])
        high_priority_signal = trans_time[:i]
        LHS = block_time
        while 1:
            RHS = block_time
            for j in range(len(high_priority_signal)):
                RHS += math.ceil((one_bit_trans +
LHS)/period_time[j])*high_priority_signal[j]
            if (RHS == LHS) & (i != (signal_num-1)) :
                # print("signal: %s response time: %s"%(i, (RHS +
trans_time[i])))
                break
            elif (RHS == LHS) & (i == (signal_num-1)):
                # print("signal: %s response time: %s"%(i, (RHS +
trans_time[i])))
                return RHS + trans_time[i]
            elif RHS >= LHS:
                LHS = RHS
            else:
                print("error in message %s"% (i))
                break
def get_neighbor(message_property):
    message_property =
message_property[np.random.choice(range(signal_num), signal_num,
replace=False)]
    return message_property

def accept_prob(delta_cost, temperature):
    if delta_cost < 0:
        return 1
    else:
        accept_rate = np.exp(-(delta_cost) / temperature)
        return accept_rate

for idx, line in enumerate(open("input.dat", 'r')):
    item = line.rstrip()
    split_item = item.split()
    if idx == 0:
        signal_num = int(split_item[0])
        trans_time = np.zeros(signal_num)
        period_time = np.zeros(signal_num)
    elif idx == 1:
        one_bit_trans = float(split_item[0])
    else:
        trans_time[int(split_item[0])] = float(split_item[1])
        period_time[int(split_item[0])] = float(split_item[2])

message_property = np.c_[np.arange(signal_num), trans_time,
period_time]
temperature = 1
max_step = 200

optimum_state = message_property
optimum_cost = calculate_response(len(message_property), one_bit_trans,
message_property[:, 1], message_property[:, 2])
print(temperature, optimum_cost)
for step in range(max_step):
    frac = step/max_step
    T = temperature * (1 - frac)
    new_state = get_neighbor(optimum_state)
    new_cost = calculate_response(len(new_state), one_bit_trans,
new_state[:, 1], new_state[:, 2])
    if accept_prob(new_cost-optimum_cost, T) > np.random.rand():
        optimum_state, optimum_cost = new_state, new_cost
        print(T,optimum_cost)
print(optimum_state[:, 1:])
print("optimum response time: %s"%(optimum_cost))
```