Come dare il resto (`conio1` [40 punti])

Comprendiamo perchè le zecche europee hanno messo in circolazione monete e banconote dei seguenti tagli (valori monetari):

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
1€	2€	5€	10€	20€	50€	100€	200€	500€	1000€

Una cassiera deve dare un resto di 98€ e, pur avendo a disposizione quanti pezzi vuole dei vari tagli della zecca, affronta il problema di usare il minimo numero di pezzi possibile nella composizione della somma spettante. In qualche modo decide di saldare come segue:

$$98€ = 50€ + 20€ + 20€ + 10€ + 5€ + 2€ + 1€$$

cavandosela maneggiando solo 7 pezzi e minimizzandone la circolazione ed usura oltrechè spedire le operazioni di cassa. Identifica l'algoritmo che computa come saldare una somma minimizzando i pezzi di denaro coinvolti nella transazione.

Input

Si legga l'input da `stdin`. La prima riga contiene T, il numero di testcase (istanze) da risolvere. Ogni istanza consta di un singolo numero naturale S. Per T volte, il server scrive una riga contenente il solo numero S ed attende la tua risposta prima di emettere la riga successiva con la prossima istanza o terminare. Inclusa la prima, il server scriverà un totale di $T + 1$ righe.

Output

Per ciascuna istanza, prima di leggere l'istanza successiva o per concludere la sessione col server, scrivi su `stdout` il tuo output così strutturato su due righe:

N									
(int)									
X_1	X_2	X_5	X_{10}	X_{20}	X_{50}	X_{100}	X_{200}	X_{500}	X_{1000}
(int)	(int)	(int)	(int)	(int)	(int)	(int)	(int)	(int)	(int)

spiegazione: nella prima riga si scrive il minimo numero di pezzi necessario per comporre la somma S, e nella seconda si specificano nel dettaglio quanti pezzi impiegare di ciascuna taglia. Si richiede di non sbagliarsi col resto ($\sum_{i=1}^{D} X_i \cdot T_i = S$) e di minimizzare il numero N di pezzi che passano di mano.

suggerimento: se non sei pratico di programmazione, comincia col calcolare N. A ciascuna riga è associato un diverso goal, e non serve che sia corretta la risposta sull'altra riga per ricevere i punti di quel goal (si deve comunque rispettare il formato di input e output per non far saltare la comunicazione col server). Se non vale $N = \sum_{i=1}^{D} X_i$ allora almeno uno dei due goal non è stato raggiunto.

Esempio di Input/Output

Input da `stdin`

```
4
98
324
3472
3456789
```

Output su `stdout`

```
6
1 1 1 0 2 1 0 0 0 0
5
0 2 0 0 1 0 1 1 0 0
8
0 1 0 0 1 1 0 2 0 3
1464
0 2 1 1 1 1 0 1 1 1456
```

Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 1 secondo.

I testcase sono raggruppati nei seguenti subtask.

1. **[8 pts← 4 istanze da 1 + 1 punti] `esempi_testo`:** i quattro esempi del testo
2. **[10 pts← 5 istanze da 1 + 1 punti] `small`:** $S \leq 30€$
3. **[8 pts← 4 istanze da 1 + 1 punti] `medium`:** $S \leq 300€$
4. **[6 pts← 3 istanze da 1 + 1 punti] `big`:** $S \leq 10.000€$
5. **[4 pts← 2 istanze da 1 + 1 punti] `large`:** $S \leq 1.000.000€$
6. **[4 pts← 2 istanze da 1 + 1 punti] `extra_large`:** $S \leq 10.000.000€$

In generale, quando si richiede la valutazione di un subtask vengono valutati anche i subtask che li precedono, ma si evita di avventurarsi in subtask successivi fuori dalla portata del tuo programma che potrebbe andare in crash o comportare tempi lunghi per ottenere la valutazione completa della sottomissione. Ad esempio, chiamando[1, 2] :

```
rtal -s <URL>  connect -x <token> -a size=medium
        conio1  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `small`, `medium`.

Il valore di default per l'argomento `size` è `extra_large` che include tutti i testcase.

[1] <URL> server esame: `wss://ta.di.univr.it/esame`

[2] <URL> server esercitazioni e simula-prove: `wss://ta.di.univr.it/algo`