Gondola

Mao-Kong Gondola is a famous attraction in Taipei. The gondola system consists of a circular rail, a single station, and n gondolas numbered consecutively from 1 to n running around the rail in a fixed direction. Initially after gondola i passes the station, the next gondola to pass the station will be gondola $i+1$ if $i < n$, or gondola 1 if $i = n$.

Gondolas may break down. Luckily we have an infinite supply of spare gondolas, which are numbered $n+1$, $n+2$, and so on. When a gondola breaks down we replace it (in the same position on the track) with the first available spare gondola, that is, the one with the lowest number. For example, if there are five gondolas and gondola 1 breaks down, then we will replace it with gondola 6.

You like to stand at the station and watch the gondolas as they pass by. A *gondola sequence* is a series of n numbers of gondolas that pass the station. It is possible that one or more gondolas broke down (and were replaced) before you arrived, but none of the gondolas break down while you are watching.

Note that the same configuration of gondolas on the rail can give multiple gondola sequences, depending on which gondola passes first when you arrive at the station. For example, if none of the gondolas have broken down then both (2, 3, 4, 5, 1) and (4, 5, 1, 2, 3) are possible gondola sequences, but (4, 3, 2, 5, 1) is not (because the gondolas appear in the wrong order).

If gondola 1 breaks down, then we might now observe the gondola sequence (4, 5, 6, 2, 3). If gondola 4 breaks down next, we replace it with gondola 7 and we might observe the gondola sequence (6, 2, 3, 7, 5). If gondola 7 breaks down after this, we replace it with gondola 8 and we may now observe the gondola sequence (3, 8, 5, 6, 2).

broken gondola	new gondola	possible gondola sequence
1	6	(4, 5, 6, 2, 3)
4	7	(6, 2, 3, 7, 5)
7	8	(3, 8, 5, 6, 2)

A *replacement sequence* is a sequence consisting of the numbers of the gondolas that have broken down, in the order in which they break down. In the previous example the replacement sequence is (1, 4, 7). A replacement sequence r *produces* a gondola sequence g if, after gondolas break down according to the replacement sequence r, the gondola sequence g may be observed.

Überprüfen der Gondelsequenz

Bei den ersten drei Subtasks musst du überprüfen, ob eine Eingabesequenz einer möglichen Gondelsequenz entspricht. Die untestehende Tabelle zeigt Beispiele von möglichen und von nicht möglichen Gondelsequenzen. Implementiere eine Funktion `valid`.

- `valid(n, inputSeq)`
  - `n`: die Länge der Eingabesequenz.
  - `inputSeq`: Array der Länge n; `inputSeq[i]` entspricht Element i der Eingabesequenz, für $0 \leq i \leq n-1$.
  - Entspricht die Eingabesequenz einer Gondelsequenz, gibt die Funktion 1 zurück, ansonsten 0.

Subtasks 1, 2, 3

Subtask	Punkte	n	`inputSeq`
1	5	$n \leq 100$	enthält jede Zahl von 1 bis n exakt einmal
2	5	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq n$
3	10	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 250,000$

Beispiele

Subtask	`inputSeq`	Rückgabewert	Bemerkung
1	(1, 2, 3, 4, 5, 6, 7)	1	
1	(3, 4, 5, 6, 1, 2)	1	
1	(1, 5, 3, 4, 2, 7, 6)	0	1 kann nicht genau vor 5 erscheinen
1	(4, 3, 2, 1)	0	4 kann nicht genau vor 3 erscheinen
2	(1, 2, 3, 4, 5, 6, 5)	0	zwei Gondeln mit der Nummer 5
3	(2, 3, 4, 9, 6, 7, 1)	1	Ersatzsequenz (5, 8)
3	(10, 4, 3, 11, 12)	0	4 kann nicht genau vor 3 erscheinen

"Replacement"-Sequenz

In den nächsten drei Subtasks konstruierst du jeweils eine mögliche "Replacement"-Sequenz, die eine gegebene Gondelsequenz erzeugt. Jede entsprechende "Replacement"-Sequenz wird akzeptiert werden. Implementiere eine Funktion `replacement`.

- `replacement(n, gondolaSeq, replacementSeq)`
  - `n`: die Länge der Gondelsequenz.
  - `gondolaSeq`: Array der Länge n; `gondolaSeq` ist garantiert eine Gondelsequenz, und `gondolaSeq[i]` ist Element i der Sequenz, für $0 \leq i \leq n-1$.
  - Die Funktion soll die Länge l der "Replacement"-Sequenz zurückgeben.
  - `replacementSeq`: Array, gross genug um die "Replacement"-Sequenz zu speichern; die Rückgabe deiner Sequenz erfolgt durch Speichern von Element i deiner "Replacement"-

Sequenz in `replacementSeq[i]`, für $0 \leq i \leq l-1$.

Subtasks 4, 5, 6

Subtask	Punkte	n	`gondolaSeq`
4	5	$n \leq 100$	$1 \leq$ `gondolaSeq[i]` $\leq n+1$
5	10	$n \leq 1,000$	$1 \leq$ `gondolaSeq[i]` $\leq 5,000$
6	20	$n \leq 100,000$	$1 \leq$ `gondolaSeq[i]` $\leq 250,000$

Beispiele

Subtask	`gondolaSeq`	Rückgabewert	`replacementSeq`
4	(3, 1, 4)	1	(2)
4	(5, 1, 2, 3, 4)	0	( )
5	(2, 3, 4, 9, 6, 7, 1)	2	(5, 8)

Anzahl der "Replacement"-Sequenzen

In den nächsten vier Subtasks ermittelst du jeweils die Anzahl der möglichen "Replacement"-Sequenzen für eine gegebene Sequenz (ob sie eine Gondelsequenz darstellt oder nicht), Modulo **1,000,000,009**. Implementiere eine Funktion `countReplacement`.

- `countReplacement(n, inputSeq)`
  - `n`: die Länge der Gondelsequenz.
  - `inputSeq`: Array der Länge n; `inputSeq[i]` ist Element i der Eingabesequenz, für $0 \leq i \leq n-1$.
  - Ist die Eingabesequenz eine Gondelsequenz, ermittle die - eventuell extrem grosse - Anzahl der "Replacement"-Sequenzen, welche diese Gondelsequenz erzeugt, *und gib diese Anzahl Modulo **1,000,000,009** zurück*. Ist die Eingabesequenz keine Gondelsequenz, gibt die Funktion 0 zurück. Ist die Eingabesequenz eine Gondelsequenz in der keine Gondeln defekt sind, gibt die Funktion 1 zurück.

Subtasks 7, 8, 9, 10

Subtask	Punkte	n	`inputSeq`
7	5	$4 \leq n \leq 50$	$1 \leq$ `inputSeq[i]` $\leq n+3$
8	15	$4 \leq n \leq 50$	$1 \leq$ `inputSeq[i]` ≤ 100, und mindestens $n-3$ der ursprünglichen Gondeln $1, \ldots, n$ sind nicht defekt.
9	15	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 250,000$
10	10	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 1,000,000,000$

Beispiele

Subtask	`inputSeq`	Rückgabewert	"Replacement"-Sequenz
7	(1, 2, 7, 6)	2	(3, 4, 5) or (4, 5, 3)
8	(2, 3, 4, 12, 6, 7, 1)	1	(5, 8, 9, 10, 11)
9	(4, 7, 4, 7)	0	`inputSeq` ist keine Gondelsequenz
10	(3, 4)	2	(1, 2) or (2, 1)

Implementierungsdetails

Gib exakt eine Datei ab. Gib ihr einen der Namen `gondola.c`, `gondola.cpp` oder `gondola.pas`. Diese Datei alle drei oben beschriebenen Subprograms implementieren (sogar, wenn du planst, nur einige Subtasks zu implementieren). Gebrauche eine der folgenden Signaturen. Du musst auch die Headerdatei `gondola.h` in die C/C++ Implementation einfügen.

C/C++ Programme

```
int valid(int n, int inputSeq[]);
int replacement(int n, int gondolaSeq[], int replacementSeq[]);
int countReplacement(int n, int inputSeq[]);
```

Pascal Programme

```
function valid(n: longint; inputSeq: array of longint): integer;
function replacement(n: longint; gondolaSeq: array of longint;
var replacementSeq: array of longint): longint;
function countReplacement(n: longint; inputSeq: array of longint):
longint;
```

Sample-Grader

Der Sample-Grader liest die Eingabe im folgenden Format:

- Zeile 1: `T`, die Nummer der Subtask die dein Programm zu lösen versucht ($\mathbf{1 \leq T \leq 10}$).
- Zeile 2: `n`, die Länge der Eingabesequenz.
- Zeile 3: Wenn `T` 4, 5, oder 6 ist, enthält diese Zeile `inputSeq[0]`, ..., `inputSeq[n-1]`. Ansonsten enthält sie `gondolaSeq[0]`, ..., `gondolaSeq[n-1]`.