Permutation

The Pharaohs use the relative movement and gravity of planets to accelerate their spaceships. Suppose a spaceship will pass by n planets with orbital speeds $p[0], p[1], \ldots, p[n-1]$ in order. For each planet, the Pharaohs scientists can choose whether to accelerate the spaceship using this planet or not. To save energy, after accelerating by a planet with orbital speed $p[i]$, the spaceship cannot be accelerated using any planet with orbital speed $p[j] < p[i]$. In other words, the chosen planets form an **increasing subsequence** of $p[0], p[1], \ldots, p[n-1]$. A subsequence of p is a sequence that's derived from p by deleting zero or more elements of p. For example $[0]$, $[]$, $[0, 2]$, and $[0, 1, 2]$ are subsequences of $[0, 1, 2]$, but $[2, 1]$ is not.

The scientists have identified that there are a total of k different ways a set of planets can be chosen to accelerate the spaceship, but they have lost their record of all the orbital speeds (even the value of n). However, they remember that $(p[0], p[1], \ldots, p[n-1])$ is a permutation of $0, 1, \ldots, n-1$. A permutation is a sequence containing each integer from 0 to $n-1$ exactly once. Your task is to find one possible permutation $p[0], p[1], \ldots, p[n-1]$ of sufficiently small length.

You need to solve the problem for q different spaceships. For each spaceship i, you get an integer k_i, representing the number of different ways a set of planets can be chosen to accelerate the spaceship. Your task is to find a sequence of orbital speeds with a small enough length n_i such that there are exactly k_i ways a subsequence of planets with increasing orbital speeds can be chosen.

Implementation details

You should implement the following procedure:

```
int[] construct_permutation(int64 k)
```

- k: is the desired number of increasing subsequences.
- This procedure should return an array of n elements where each element is between 0 and $n-1$ inclusive.
- The returned array must be a valid permutation having exactly k increasing subsequences.
- This procedure is called a total q of times. Each of these calls should be treated as a separate scenario.

Constraints

- $1 \le q \le 100$
- $2 \le k_i \le 10^{18}$ (for all $0 \le i \le q-1$)

Subtasks

1. (10 points) $2 \le k_i \le 90$ (for all $0 \le i \le q-1$). If all permutations you used have length at most 90 and are correct, you receive 10 points, otherwise 0.
2. (90 points) No additional constraints. For this subtask, let m be the maximum permutation length you used in any scenario. Then, your score is calculated according to the following table:

Condition	Score
$m \le 90$	90
$90 < m \le 120$	$90 - \frac{(m-90)}{3}$
$120 < m \le 5000$	$80 - \dfrac{(m-120)}{65}$
$m > 5000$	0

Example

Example 1

Consider the following call:

```
construct_permutation(3)
```

This procedure should return a permutation with exactly 3 increasing subsequences. A possible answer is $[1, 0]$, which has $[]$ (empty subsequence), $[0]$ and $[1]$ as increasing subsequences.

Example 2

Consider the following call:

```
construct_permutation(8)
```

This procedure should return a permutation with exactly 8 increasing subsequences. A possible answer is $[0, 1, 2]$.

Sample grader

The sample grader reads the input in the following format:

- line 1: q
- line $2+i$ ($0 \leq i \leq q-1$): k_i

The sample grader prints a single line for each k_i containing the return value of `construct_permutation`, or an error message if one has occurred.