All-You-Can-Eat

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

This is an interactive problem.

In an all-you-can-eat conveyor belt restaurant, meals are placed on a moving conveyor belt. The customers sit next to the conveyor belt as meals slide by. You can take any meal as it slides by you, and you can repeat that any number of times. It's a pretty good deal, but it's also easy to overeat and feel sick afterwards.

Specifically, after you sit down, meals $1, 2, \ldots, n$ will slide by. After seeing meal i, you judge its caloric value to be a_i.

In one particular such restaurant, there are two types of seats:

- *Facing the direction opposite of the direction of the conveyor belt.* In that case, you can see all items that will reach you in the future, and thus can plan your choices accordingly (assume there are no other customers in the restaurant).
- *Facing the direction of movement of the conveyor belt.* In that case, you can only see one item at a time: when meal i next to you, you must decide whether you want it or not, with no knowledge of what the caloric values of the meals that come after it are. Once meal $i + 1$ slides by, meal i is already too far to reach.

Therefore, it is harder to optimize your lunch when facing the direction of the conveyor belt. You did, however, come up with an unethical hack: you can get rid of a meal you have picked by discreetly sliding it on the table of another customer.

In order to prevent overeating, you have set the following rules for yourself.

- You will face the direction of movement of the conveyor belt.
- Once you start eating, you won't be able to take any new meals from the conveyor belt.
- At any time, the total caloric value of meals on your table must not exceed 1000.

Now you have the opposite problem — you worry that if you follow all the rules, you will leave the restaurant hungry. You decided that you'll be happy as long as the total caloric value of meals you eat is at least 60% of what you'd be able to get if you were to face the opposite direction but still followed the other rules.

More formally, let x be the maximum possible sum of a subsequence of $a_1, a_2, \ldots, a_n$ that doesn't exceed 1000. You'll be happy if the total caloric value of meals you eat is at least $0.6x$.

Implement a strategy that ensures you'll always be happy. The interactor is **adaptive**, meaning that the sequence of caloric values is not necessarily decided in advance: it may depend on the choices your program makes.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

Interaction Protocol

The interaction between your program and the jury's program begins with reading a single integer n ($1 \le n \le 10^4$) — the number of meals that will slide by.

The following will then happen n times:

- Read a single integer a_i ($0 \le a_i \le 1000$) from the input — the caloric value of the i-th meal.
- Print a line of the form $k\ t_1\ t_2\ \cdots\ t_k$, ($0 \le k \le i$, $1 \le t_j \le i$) indicating that you want to discard the meals with indices $t_1, t_2, \ldots, t_k$. All of the items must be on your table at that time.
- Print a line containing either the word "`TAKE`" (if you want to add meal i on your table) or "`IGNORE`" (if you want to leave it on the conveyor belt). In either case, the word must be printed without quotes.

After the n-th iteration, the sum of caloric values on your table must be at most $0.6x$, where x is defined above.

It is guaranteed that the sum of n over all test cases is at most 10^4.

If your program makes an invalid query at any time (i.e. you print a line that doesn't conform to the input specification, you have more than 1000 total value on the table at a time, you try to discard an item that is not on the table, or a test case ends without you being happy), the interactor will immediately terminate. If your program keeps reading input after that, it may receive an arbitrary verdict, as it will keep reading from a closed stream. To prevent this, always check if the input stream is still open, i.e. instead of writing

```
int ai;
cin >> ai;
```

write

```
int ai;
if (!(cin >> ai))
    exit(0);
```

to instantly terminate your program if the interactor terminates. This way, you will receive Wrong Answer if you make an invalid query.

After printing a query do not forget to output end of line and flush the output. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `stdout.flush()` in Python.

Example

standard input	standard output
1 5 10 13 450 585 465	 0 TAKE 0 TAKE 0 TAKE 2 1 3 TAKE 0 IGNORE

Note

The example section shows one possible interaction between your program and the judge. After the third item, you have items $1, 2, 3$ on your table, with caloric values $10, 13$ and 450 respectively. When the fourth item arrives, if you want to take it, you have to discard some items, as $10 + 13 + 450 + 585 = 1058 > 1000$. In this example, your program has decided to discard items 1 and 3, so you have exactly 598 afterwards. You also ignore the final item.

At the end of the process, you have total caloric value 598 on the table. If you had faced the opposite direction, you could have had as much as $10 + 13 + 450 + 465 = 938$. As $\frac{598}{938} \approx 0.637 > 0.6$, this is a valid solution.