# Codeforces Round #744 (Div. 3)

## A. Casimir's String Solitaire

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Casimir has a string $s$ which consists of capital Latin letters 'A', 'B', and 'C' only. Each turn he can choose to do one of the two following actions:

- he can either erase exactly one letter 'A' **and** exactly one letter 'B' from arbitrary places of the string (these letters don't have to be adjacent);
- or he can erase exactly one letter 'B' **and** exactly one letter 'C' from arbitrary places in the string (these letters don't have to be adjacent).

Therefore, each turn the length of the string is decreased exactly by $2$. All turns are independent so for each turn, Casimir can choose any of two possible actions.

For example, with $s$ = "ABCABC" he can obtain a string $s$ = "ACBC" in one turn (by erasing the first occurrence of 'B' and the second occurrence of 'A'). There are also many other options for a turn aside from this particular example.

For a given string $s$ determine whether there is a sequence of actions leading to an empty string. In other words, Casimir's goal is to erase all letters from the string. Is there a way to do this?

### Input

The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

Each test case is described by one string $s$, for which you need to determine if it can be fully erased by some sequence of turns. The string $s$ consists of capital letters 'A', 'B', 'C' and has a length from $1$ to $50$ letters, inclusive.

### Output

Print $t$ lines, each line containing the answer to the corresponding test case. The answer to a test case should be YES if there is a way to fully erase the corresponding string and NO otherwise.

You may print every letter in any case you want (so, for example, the strings yEs, yes, Yes, and YES will all be recognized as positive answers).

**Example**

| input |
| --- |
| 6<br>ABACAB<br>ABBA<br>AC<br>ABC<br>CABCBB<br>BCBCBCBCBCBCBCBC |
| **output** |
| NO<br>YES<br>NO<br>NO<br>YES<br>YES |

## B. Shifting Sort

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

The new generation external memory contains an array of integers $a[1 \ldots n] = [a_1, a_2, \ldots, a_n]$.

This type of memory does not support changing the value of an arbitrary element. Instead, it allows you to cut out any segment of the given array, *cyclically shift* (rotate) it by any offset and insert it back into the same place.

Technically, each cyclic shift consists of two consecutive actions:

1. You may select arbitrary indices $l$ and $r$ ($1 \le l < r \le n$) as the boundaries of the segment.

2. Then you replace the segment $a[l \dots r]$ with it's *cyclic shift* to the **left** by an arbitrary offset $d$. The concept of a *cyclic shift* can be also explained by following relations: the sequence $[1, 4, 1, 3]$ is a cyclic shift of the sequence $[3, 1, 4, 1]$ to the left by the offset $1$ and the sequence $[4, 1, 3, 1]$ is a cyclic shift of the sequence $[3, 1, 4, 1]$ to the left by the offset $2$.

For example, if $a = [1, 3, 2, 8, 5]$, then choosing $l = 2$, $r = 4$ and $d = 2$ yields a segment $a[2 \dots 4] = [3, 2, 8]$. This segment is then shifted by the offset $d = 2$ to the **left**, and you get a segment $[8, 3, 2]$ which then takes the place of of the original elements of the segment. In the end you get $a = [1, 8, 3, 2, 5]$.

Sort the given array $a$ using no more than $n$ cyclic shifts of any of its segments. Note that you don't need to minimize the number of cyclic shifts. Any method that requires $n$ or less cyclic shifts will be accepted.

### Input
The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

The next $2t$ lines contain the descriptions of the test cases.

The first line of each test case description contains an integer $n$ ($2 \le n \le 50$) — the length of the array. The second line consists of space-separated elements of the array $a_i$ ($-10^9 \le a_i \le 10^9$). Elements of array $a$ may repeat and don't have to be unique.

### Output
Print $t$ answers to all input test cases.

The first line of the answer of each test case should contain an integer $k$ ($0 \le k \le n$) — the number of actions to sort the array. The next $k$ lines should contain descriptions of the actions formatted as "$l$ $r$ $d$" (without quotes) where $l$ and $r$ ($1 \le l < r \le n$) are the boundaries of the segment being shifted, while $d$ ($1 \le d \le r - l$) is the offset value. Please remember that only the cyclic shifts **to the left** are considered so the chosen segment will be shifted by the offset $d$ to the *to the left*.

Note that you are not required to find the minimum number of cyclic shifts needed for sorting. Any sorting method where the number of shifts does not exceed $n$ will be accepted.

If the given array $a$ is already sorted, one of the possible answers is $k = 0$ and an empty sequence of cyclic shifts.

If there are several possible answers, you may print any of them.

**Example**

| input |
|---|
| 4<br>2<br>2 1<br>3<br>1 2 1<br>4<br>2 4 1 3<br>5<br>2 5 1 4 3 |
| **output** |
| 1<br>1 2 1<br>1<br>1 3 2<br>3<br>2 4 1<br>2 3 1<br>1 3 2<br>4<br>2 4 2<br>1 5 3<br>1 2 1<br>1 3 1 |

### Note
Explanation of the fourth data set in the example:

1. The segment $a[2 \dots 4]$ is selected and is shifted to the left by $2$: $[2, 5, 1, 4, 3] \longrightarrow [2, 4, 5, 1, 3]$
2. The segment $a[1 \dots 5]$ is then selected and is shifted to the left by $3$: $[2, 4, 5, 1, 3] \longrightarrow [1, 3, 2, 4, 5]$
3. After that the segment $a[1 \dots 2]$ is selected and is shifted to the left by $1$: $[1, 3, 2, 4, 5] \longrightarrow [3, 1, 2, 4, 5]$
4. And in the end the segment $a[1 \dots 3]$ is selected and is shifted to the left by $1$: $[3, 1, 2, 4, 5] \longrightarrow [1, 2, 3, 4, 5]$

## C. Ticks

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

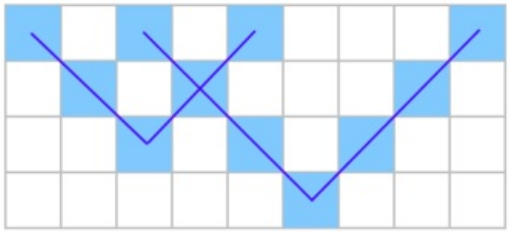
Casimir has a rectangular piece of paper with a checkered field of size $n \times m$. Initially, all cells of the field are white.

Let us denote the cell with coordinates $i$ vertically and $j$ horizontally by $(i, j)$. The upper left cell will be referred to as $(1, 1)$ and the lower right cell as $(n, m)$.

Casimir draws *ticks* of different sizes on the field. A tick of size $d$ ($d > 0$) with its center in cell $(i, j)$ is drawn as follows:

1. First, the center cell $(i, j)$ is painted black.
2. Then exactly $d$ cells on the top-left diagonally to the center and exactly $d$ cells on the top-right diagonally to the center are also painted black.
3. That is all the cells with coordinates $(i - h, j \pm h)$ for all $h$ between $0$ and $d$ are painted. In particular, a tick consists of $2d + 1$ black cells.

An already painted cell will remain black if painted again. Below you can find an example of the $4 \times 9$ box, with two ticks of sizes $2$ and $3$.



You are given a description of a checkered field of size $n \times m$. Casimir claims that this field came about after he drew some (possibly $0$) ticks on it. The ticks could be of different sizes, but the size of each tick is at least $k$ (that is, $d \geq k$ for all the ticks).

Determine whether this field can indeed be obtained by drawing some (possibly none) ticks of sizes $d \geq k$ or not.

## Input

The first line contains an integer $t$ ($1 \leq t \leq 100$) — the number test cases.

The following lines contain the descriptions of the test cases.

The first line of the test case description contains the integers $n$, $m$, and $k$ ($1 \leq k \leq n \leq 10$; $1 \leq m \leq 19$) — the field size and the minimum size of the ticks that Casimir drew. The following $n$ lines describe the field: each line consists of $m$ characters either being '`.`' if the corresponding cell is not yet painted or '*' otherwise.

## Output

Print $t$ lines, each line containing the answer to the corresponding test case. The answer to a test case should be YES if the given field can be obtained by drawing ticks of at least the given size and NO otherwise.

You may print every letter in any case you want (so, for example, the strings yEs, yes, Yes, and YES will all be recognized as positive answers).

**Example**

| input |
| --- |
| 8<br>2 3 1<br>*.*<br>...<br>4 9 2<br>*.*.*...*<br>.*.*...*.<br>..*.*.*..<br>.....*...<br>4 4 1<br>*.*.<br>****<br>.**.<br>....<br>5 5 1<br>.....<br>*...*<br>.*.*.<br>..*.*<br>...*.<br>5 5 2<br>.....<br>*...*<br>.*.*.<br>..*.*<br>...*.<br>4 7 1<br>*.....*<br>.....*.<br>..*.*..<br>...*...<br>3 3 1<br>***<br>***<br>***<br>3 5 1<br>*...*<br>.***.<br>.**.. |
| **output** |
| NO |

```
YES
YES
YES
NO
NO
NO
NO
```

### Note

The first sample test case consists of two asterisks neither of which can be independent ticks since ticks of size $0$ don't exist.

The second sample test case is already described in the statement (check the picture in the statement). This field can be obtained by drawing ticks of sizes $2$ and $3$, as shown in the figure.

The field in the third sample test case corresponds to three ticks of size $1$. Their center cells are marked with blue, red and green colors:

```
*.*.
****
.**.
....
```

The field in the fourth sample test case could have been obtained by drawing two ticks of sizes $1$ and $2$. Their vertices are marked below with blue and red colors respectively:

```
.....
*...*
.*.*.
..*.*
...*.
```

The field in the fifth sample test case can not be obtained because $k = 2$, and the last asterisk in the fourth row from the top with coordinates $(4, 5)$ can only be a part of a tick of size $1$.

The field in the sixth sample test case can not be obtained because the top left asterisk $(1, 1)$ can't be an independent tick, since the sizes of the ticks must be positive, and cannot be part of a tick with the center cell in the last row, since it is separated from it by a gap (a point, '.') in $(2, 2)$.

In the seventh sample test case, similarly, the field can not be obtained by the described process because the asterisks with coordinates $(1, 2)$ (second cell in the first row), $(3, 1)$ and $(3, 3)$ (leftmost and rightmost cells in the bottom) can not be parts of any ticks.

## D. Productive Meeting

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

An important meeting is to be held and there are exactly $n$ people invited. At any moment, any two people can step back and talk in private. The same two people can talk several (as many as they want) times per meeting.

Each person has limited *sociability*. The sociability of the $i$-th person is a non-negative integer $a_i$. This means that after exactly $a_i$ talks this person leaves the meeting (and does not talk to anyone else anymore). If $a_i = 0$, the $i$-th person leaves the meeting immediately after it starts.

A meeting is considered most *productive* if the maximum possible number of talks took place during it.

You are given an array of sociability $a$, determine which people should talk to each other so that the total number of talks is as large as possible.

### Input

The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

The next $2t$ lines contain descriptions of the test cases.

The first line of each test case description contains an integer $n$ ($2 \le n \le 2 \cdot 10^5$) —the number of people in the meeting. The

second line consists of $n$ space-separated integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 2 \cdot 10^5$) — the sociability parameters of all people.

It is guaranteed that the sum of $n$ over all test cases does not exceed $2 \cdot 10^5$. It is also guaranteed that the sum of all $a_i$ (over all test cases and all $i$) does not exceed $2 \cdot 10^5$.

**Output**

Print $t$ answers to all test cases.

On the first line of each answer print the number $k$ — the maximum number of talks possible in a meeting.

On each of the next $k$ lines print two integers $i$ and $j$ ($1 \le i, j \le n$ and $i \ne j$) — the numbers of people who will have another talk.

If there are several possible answers, you may print any of them.

**Example**

| input |
| --- |
| 8<br>2<br>2 3<br>3<br>1 2 3<br>4<br>1 2 3 4<br>3<br>0 0 2<br>2<br>6 2<br>3<br>0 0 2<br>5<br>8 2 0 1 1<br>5<br>0 1 0 0 6 |

| output |
| --- |
| 2<br>1 2<br>1 2<br>3<br>1 3<br>2 3<br>2 3<br>5<br>1 3<br>2 4<br>2 4<br>3 4<br>3 4<br>0<br>2<br>1 2<br>1 2<br>0<br>4<br>1 2<br>1 5<br>1 4<br>1 2<br>1<br>5 2 |

# E1. Permutation Minimization by Deque

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

*In fact, the problems E1 and E2 do not have much in common. You should probably think of them as two separate problems.*

A permutation $p$ of size $n$ is given. A *permutation* of size $n$ is an array of size $n$ in which each integer from $1$ to $n$ occurs exactly once. For example, $[1, 4, 3, 2]$ and $[4, 2, 1, 3]$ are correct permutations while $[1, 2, 4]$ and $[1, 2, 2]$ are not.

Let us consider an empty deque (double-ended queue). A deque is a data structure that supports adding elements to both the beginning and the end. So, if there are elements $[1, 5, 2]$ currently in the deque, adding an element $4$ to the beginning will produce the sequence $[4, 1, 5, 2]$, and adding same element to the end will produce $[1, 5, 2, 4]$.

The elements of the permutation are sequentially added to the initially empty deque, starting with $p_1$ and finishing with $p_n$. Before adding each element to the deque, you may choose whether to add it to the beginning or the end.

For example, if we consider a permutation $p = [3, 1, 2, 4]$, one of the possible sequences of actions looks like this:

1. add $3$ to the end of the deque: deque has a sequence $[3]$ in it;

| | | |
|---|---|---|
| 2. | add $1$ to the beginning of the deque: | deque has a sequence $[1, 3]$ in it; |
| 3. | add $2$ to the end of the deque: | deque has a sequence $[1, 3, 2]$ in it; |
| 4. | add $4$ to the end of the deque: | deque has a sequence $[1, 3, 2, 4]$ in it; |

Find the lexicographically smallest possible sequence of elements in the deque after the entire permutation has been processed.

A sequence $[x_1, x_2, \ldots, x_n]$ is *lexicographically smaller* than the sequence $[y_1, y_2, \ldots, y_n]$ if there exists such $i \leq n$ that $x_1 = y_1$, $x_2 = y_2$, $\ldots$, $x_{i-1} = y_{i-1}$ and $x_i < y_i$. In other words, if the sequences $x$ and $y$ have some (possibly empty) matching prefix, and the next element of the sequence $x$ is strictly smaller than the corresponding element of the sequence $y$. For example, the sequence $[1, 3, 2, 4]$ is smaller than the sequence $[1, 3, 4, 2]$ because after the two matching elements $[1, 3]$ in the start the first sequence has an element $2$ which is smaller than the corresponding element $4$ in the second sequence.

### Input
The first line contains an integer $t$ ($1 \leq t \leq 1000$) — the number of test cases.

The next $2t$ lines contain descriptions of the test cases.

The first line of each test case description contains an integer $n$ ($1 \leq n \leq 2 \cdot 10^5$) — permutation size. The second line of the description contains $n$ space-separated integers $p_i$ ($1 \leq p_i \leq n$; all $p_i$ are all unique) — elements of the permutation.

It is guaranteed that the sum of $n$ over all test cases does not exceed $2 \cdot 10^5$.

### Output
Print $t$ lines, each line containing the answer to the corresponding test case. The answer to a test case should contain $n$ space-separated integer numbers — the elements of the lexicographically smallest permutation that is possible to find in the deque after executing the described algorithm.

**Example**

| input |
|---|
| 5<br>4<br>3 1 2 4<br>3<br>3 2 1<br>3<br>3 1 2<br>2<br>1 2<br>2<br>2 1 |
| **output** |
| 1 3 2 4<br>1 2 3<br>1 3 2<br>1 2<br>1 2 |

### Note
One of the ways to get a lexicographically smallest permutation $[1, 3, 2, 4]$ from the permutation $[3, 1, 2, 4]$ (the first sample test case) is described in the problem statement.

## E2. Array Optimization by Deque

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

*In fact, the problems E1 and E2 do not have much in common. You should probably think of them as two separate problems.*

You are given an integer array $a[1 \ldots n] = [a_1, a_2, \ldots, a_n]$.

Let us consider an empty deque (double-ended queue). A deque is a data structure that supports adding elements to both the beginning and the end. So, if there are elements $[3, 4, 4]$ currently in the deque, adding an element $1$ to the beginning will produce the sequence $[1, 3, 4, 4]$, and adding the same element to the end will produce $[3, 4, 4, 1]$.

The elements of the array are sequentially added to the initially empty deque, starting with $a_1$ and finishing with $a_n$. Before adding each element to the deque, you may choose whether to add it to the beginning or to the end.

For example, if we consider an array $a = [3, 7, 5, 5]$, one of the possible sequences of actions looks like this:

| | | |
|---|---|---|
| 1. | add $3$ to the beginning of the deque: | deque has a sequence $[3]$ in it; |

| | | |
|---|---|---|
| 2. | add $7$ to the end of the deque: | deque has a sequence $[3, 7]$ in it; |
| 3. | add $5$ to the end of the deque: | deque has a sequence $[3, 7, 5]$ in it; |
| 4. | add $5$ to the beginning of the deque: | deque has a sequence $[5, 3, 7, 5]$ in it; |

Find the minimal possible number of inversions in the deque after the whole array is processed.

An *inversion* in sequence $d$ is a pair of indices $(i, j)$ such that $i < j$ and $d_i > d_j$. For example, the array $d = [5, 3, 7, 5]$ has exactly two inversions — $(1, 2)$ and $(3, 4)$, since $d_1 = 5 > 3 = d_2$ and $d_3 = 7 > 5 = d_4$.

### Input
The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

The next $2t$ lines contain descriptions of the test cases.

The first line of each test case description contains an integer $n$ ($1 \le n \le 2 \cdot 10^5$) — array size. The second line of the description contains $n$ space-separated integers $a_i$ ($-10^9 \le a_i \le 10^9$) — elements of the array.

It is guaranteed that the sum of $n$ over all test cases does not exceed $2 \cdot 10^5$.

### Output
Print $t$ lines, each line containing the answer to the corresponding test case. The answer to a test case should be a single integer — the minimal possible number of inversions in the deque after executing the described algorithm.

**Example**

| input |
|---|
| 6<br>4<br>3 7 5 5<br>3<br>3 2 1<br>3<br>3 1 2<br>4<br>-1 2 2 -1<br>4<br>4 5 1 3<br>5<br>1 3 1 3 2 |
| **output** |
| 2<br>0<br>1<br>0<br>1<br>2 |

### Note
One of the ways to get the sequence $[5, 3, 7, 5]$ in the deque, containing only two inversions, from the initial array $[3, 7, 5, 5]$ (the first sample test case) is described in the problem statement.

Also, in this example, you could get the answer of two inversions by simply putting each element of the original array at the end of the deque. In this case, the original sequence $[3, 7, 5, 5]$, also containing exactly two inversions, will be in the deque as-is.

## F. Array Stabilization (AND version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array $a[0 \ldots n-1] = [a_0, a_1, \ldots, a_{n-1}]$ of zeroes and ones only. Note that in this problem, unlike the others, the array indexes are numbered from zero, not from one.

In one step, the array $a$ is replaced by another array of length $n$ according to the following rules:

1. First, a new array $a^{\to d}$ is defined as a cyclic shift of the array $a$ to the right by $d$ cells. The elements of this array can be defined as $a^{\to d}_i = a_{(i+n-d) \bmod n}$, where $(i + n - d) \bmod n$ is the remainder of integer division of $i + n - d$ by $n$.
   It means that the whole array $a^{\to d}$ can be represented as a sequence
   $$a^{\to d} = [a_{n-d}, a_{n-d+1}, \ldots, a_{n-1}, a_0, a_1, \ldots, a_{n-d-1}]$$
2. Then each element of the array $a_i$ is replaced by $a_i \,\&\, a^{\to d}_i$, where $\&$ is a logical "AND" operator.

For example, if $a = [0, 0, 1, 1]$ and $d = 1$, then $a^{\to d} = [1, 0, 0, 1]$ and the value of $a$ after the first step will be $[0 \,\&\, 1, 0 \,\&\, 0, 1 \,\&\, 0, 1 \,\&\, 1]$, that is $[0, 0, 0, 1]$.

The process ends when the array stops changing. For a given array $a$, determine whether it will consist of only zeros at the end of the process. If yes, also find the number of steps the process will take before it finishes.

### Input
The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

The next $2t$ lines contain descriptions of the test cases.

The first line of each test case description contains two integers: $n$ ($1 \le n \le 10^6$) — array size and $d$ ($1 \le d \le n$) — cyclic shift offset. The second line of the description contains $n$ space-separated integers $a_i$ ($0 \le a_i \le 1$) — elements of the array.

It is guaranteed that the sum of $n$ over all test cases does not exceed $10^6$.

### Output
Print $t$ lines, each line containing the answer to the corresponding test case. The answer to a test case should be a single integer — the number of steps after which the array will contain only zeros for the first time. If there are still elements equal to $1$ in the array after the end of the process, print $-1$.

**Example**

| input |
| --- |
| 5<br>2 1<br>0 1<br>3 2<br>0 1 0<br>5 2<br>1 1 0 1 0<br>4 2<br>0 1 0 1<br>1 1<br>0 |
| **output** |
| 1<br>1<br>3<br>-1<br>0 |

### Note
In the third sample test case the array will change as follows:

1. At the beginning $a = [1, 1, 0, 1, 0]$, and $a^{\to 2} = [1, 0, 1, 1, 0]$. Their element-by-element "AND" is equal to
$$[1 \,\&\, 1, 1 \,\&\, 0, 0 \,\&\, 1, 1 \,\&\, 1, 0 \,\&\, 0] = [1, 0, 0, 1, 0]$$
2. Now $a = [1, 0, 0, 1, 0]$, then $a^{\to 2} = [1, 0, 1, 0, 0]$. Their element-by-element "AND" equals to
$$[1 \,\&\, 1, 0 \,\&\, 0, 0 \,\&\, 1, 1 \,\&\, 0, 0 \,\&\, 0] = [1, 0, 0, 0, 0]$$
3. And finally, when $a = [1, 0, 0, 0, 0]$ we get $a^{\to 2} = [0, 0, 1, 0, 0]$. Their element-by-element "AND" equals to
$$[1 \,\&\, 0, 0 \,\&\, 0, 0 \,\&\, 1, 0 \,\&\, 0, 0 \,\&\, 0] = [0, 0, 0, 0, 0]$$

Thus, the answer is $3$ steps.
In the fourth sample test case, the array will not change as it shifts by $2$ to the right, so each element will be calculated as $0 \,\&\, 0$ or $1 \,\&\, 1$ thus not changing its value. So the answer is $-1$, the array will never contain only zeros.

## G. Minimal Coverage

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given $n$ lengths of segments that need to be placed on an infinite axis with coordinates.

The first segment is placed on the axis so that one of its endpoints lies at the point with coordinate $0$. Let's call this endpoint the "*start*" of the first segment and let's call its "*end*" as that endpoint that is not the start.

The "start" of each following segment must coincide with the "end" of the previous one. Thus, if the length of the next segment is $d$ and the "end" of the previous one has the coordinate $x$, the segment can be placed either on the coordinates $[x - d, x]$, and then the coordinate of its "end" is $x - d$, or on the coordinates $[x, x + d]$, in which case its "end" coordinate is $x + d$.

The total *coverage* of the axis by these segments is defined as their overall union which is basically the set of points covered by at least one of the segments. It's easy to show that the coverage will also be a segment on the axis. Determine the minimal possible length of the coverage that can be obtained by placing all the segments on the axis without changing their order.

### Input

The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

The next $2t$ lines contain descriptions of the test cases.

The first line of each test case description contains an integer $n$ ($1 \le n \le 10^4$) — the number of segments. The second line of the description contains $n$ space-separated integers $a_i$ ($1 \le a_i \le 1000$) — lengths of the segments in the same order they should be placed on the axis.

It is guaranteed that the sum of $n$ over all test cases does not exceed $10^4$.

## Output

Print $t$ lines, each line containing the answer to the corresponding test case. The answer to a test case should be a single integer — the minimal possible length of the axis coverage.

**Example**

| input |
| --- |
| 6<br>2<br>1 3<br>3<br>1 2 3<br>4<br>6 2 3 9<br>4<br>6 8 4 5<br>7<br>1 2 4 6 7 7 3<br>8<br>8 6 5 1 2 2 3 6 |
| **output** |
| 3<br>3<br>9<br>9<br>7<br>8 |

## Note

In the third sample test case the segments should be arranged as follows: $[0, 6] \to [4, 6] \to [4, 7] \to [-2, 7]$. As you can see, the last segment $[-2, 7]$ covers all the previous ones, and the total length of coverage is $9$.

In the fourth sample test case the segments should be arranged as $[0, 6] \to [-2, 6] \to [-2, 2] \to [2, 7]$. The union of these segments also occupies the area $[-2, 7]$ and has the length of $9$.