Problem Z

Jinxed Betting

Algorithmen für Programmierwettbewerbe

Jonathan Klawitter
Folien von Philipp Kindermann
Problem

Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.

She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.

Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Ausgangslage

Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.
She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.
Vorgehensweise

She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.
She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.
She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.
She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Ergebnis

Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Ergebnis

<table>
<thead>
<tr>
<th>Anfang</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for **how many more matches she is guaranteed to stay in the lead in the worst case** (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.
Ein-/Ausgabe

The input consists of:

- One line with an integer \( n \) \((3 \leq n \leq 10^5)\), the number of people who place their bets.
- One line with \( n \) integers \( p_1, \ldots, p_n \) \((0 \leq p_i \leq 10^{16} \text{ for each } i)\), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.
The input consists of:

- One line with an integer \( n \) (\( 3 \leq n \leq 10^5 \)), the number of people who place their bets.
- One line with \( n \) integers \( p_1, \ldots, p_n \) (\( 0 \leq p_i \leq 10^{16} \) for each \( i \)), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.
The input consists of:

- One line with an integer $n$ ($3 \leq n \leq 10^5$), the number of people who place their bets.
- One line with $n$ integers $p_1, \ldots, p_n$ ($0 \leq p_i \leq 10^{16}$ for each $i$), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.

Input:

```
7
5 1 4 4 3 2 4
```
Ein-/Ausgabe

The input consists of:

- One line with an integer \( n \) (\( 3 \leq n \leq 10^5 \)), the number of people who place their bets.
- One line with \( n \) integers \( p_1, \ldots, p_n \) (\( 0 \leq p_i \leq 10^{16} \) for each \( i \)), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.

Output the number of matches for which Julia is guaranteed to stay in the lead.

Input:

```
7
5 1 4 4 3 2 4
```
Ein-/Ausgabe

The input consists of:

- One line with an integer $n$ ($3 \leq n \leq 10^5$), the number of people who place their bets.

- One line with $n$ integers $p_1, \ldots, p_n$ ($0 \leq p_i \leq 10^{16}$ for each $i$), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.

Output the number of matches for which Julia is guaranteed to stay in the lead.

Input:

```
7
5 1 4 4 3 2 4
```

Output:

```
2
```
The input consists of:

- One line with an integer \( n (3 \leq n \leq 10^5) \), the number of people who place their bets.
- One line with \( n \) integers \( p_1, \ldots, p_n \) \( (0 \leq p_i \leq 10^{16} \) for each \( i \)), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.

Output the number of matches for which Julia is guaranteed to stay in the lead.

Input:

```
7
5 1 4 4 3 2 4
```

Output:

```
2
```
The input consists of:

- One line with an integer \( n \) (\( 3 \leq n \leq 10^5 \)), the number of people who place their bets.

- One line with \( n \) integers \( p_1, \ldots, p_n \) (\( 0 \leq p_i \leq 10^{16} \) for each \( i \)), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia’s score in the beginning.

Output the number of matches for which Julia is guaranteed to stay in the lead.

Input:
7
5 1 4 4 3 2 4

Output:
2
Bruteforce

Probiere alles aus.
Bruteforce

Probiere alles aus.

Pro Runde:
Bruteforce

Probiere alles aus.

Pro Runde:  • $n - 1$ Teilnehmer haben eine Wahl
Brute force

Probiere alles aus.

Pro Runde: \( n - 1 \) Teilnehmer haben eine Wahl \( \Rightarrow 2^{n-1} \)
Bruteforce

Probiere alles aus.

Pro Runde:
- $n - 1$ Teilnehmer haben eine Wahl $\Rightarrow 2^{n-1}$
- 2 mögliche Ergebnisse $\Rightarrow 2$
Bruteforce

Probiere alles aus.

Pro Runde:  
• $n - 1$ Teilnehmer haben eine Wahl  ⇒ $2^{n-1}$
• 2 mögliche Ergebnisse  ⇒ 2
• Bei Gleichstand: Julia hat eine Wahl  ⇒ 2
Brute force

Probiere alles aus.

Pro Runde:  
  • $n - 1$ Teilnehmer haben eine Wahl  \( \Rightarrow 2^{n-1} \)
  • 2 mögliche Ergebnisse  \( \Rightarrow 2 \)
  • Bei Gleichstand: Julia hat eine Wahl  \( \Rightarrow 2 \)

$r$ Runden:
Bruteforce

Probiere alles aus.

Pro Runde:
- \( n - 1 \) Teilnehmer haben eine Wahl
- 2 mögliche Ergebnisse
- Bei Gleichstand: Julia hat eine Wahl

\[ 2^n - 1 \]
\[ \Rightarrow 2 \]
\[ \Rightarrow 2 \]

\( r \) Runden:
\[ 2^{r(n+1)} \]
Bruteforce

Probiere alles aus.

Pro Runde:

• $n - 1$ Teilnehmer haben eine Wahl
• 2 mögliche Ergebnisse
• Bei Gleichstand: Julia hat eine Wahl

$r$ Runden: $2^{r(n+1)}$

$n \leq 10^5, \quad r \leq 10^{16}$
Bruteforce

Probiere alles aus.

Pro Runde:  
- $n - 1$ Teilnehmer haben eine Wahl  
- 2 mögliche Ergebnisse  
- Bei Gleichstand: Julia hat eine Wahl

$r$ Runden:  
$2^{r(n+1)}$

$n \leq 10^5, \; r \leq 10^{16}$  
$\Rightarrow 2^{10^{21}}$
Worst Case

Teile in Gruppen ein:
Worst Case

Teile in Gruppen ein:

Julia
Worst Case

Teile in Gruppen ein:

Julia

Meisten Punkte
Worst Case

Teile in Gruppen ein:

Julia

Meisten Punkte

Weniger Punkte
Worst Case

Teile in Gruppen ein:

Meisten Punkte

Julia

Weniger Punkte
Worst Case

Teile in Gruppen ein:

Julia

Meisten Punkte

Weniger Punkte
Worst Case

Teile in Gruppen ein:

Julia

Meisten Punkte

Weniger Punkte

+1
Worst Case

Teile in Gruppen ein:

Julia

Meisten Punkte

Weniger Punkte

+1

$t$
Worst Case

Teile in Gruppen ein:

Meisten Punkte

Weniger Punkte

• Worst Case: Tippen genau wie \( \lceil t/2 \rceil \) Verfolger
Worst Case

Teile in Gruppen ein:

• Worst Case: Tippen genau wie ⌈t/2⌉ Verfolger
Worst Case

Teile in Gruppen ein:

Meisten Punkte

Weniger Punkte

\[ t \]

- Worst Case: Tippen genau wie \( \lceil t/2 \rceil \) Verfolger
Worst Case

Teile in Gruppen ein:

- Worst Case: Tippen genau wie \([t/2]\) Verfolger
- Teilnehmer mit gleicher Punktzahl äquivalent
Worst Case

Teile in Gruppen ein:

- Meisten Punkte
- Weniger Punkte

- Worst Case: Tippen genau wie $\lceil t/2 \rceil$ Verfolger
- Teilnehmer mit gleicher Punktzahl äquivalent
  $\Rightarrow$ eindeutige Wahl pro Runde
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]
# Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

```python
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]
# Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

```python
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]
# Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

```python
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

<table>
<thead>
<tr>
<th></th>
<th>J</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```
Naiver Algorithmus

\[
n = \text{int}(\text{input}()) \quad \# \text{ Python2: int(raw_input())}
\]

scores = [\text{int}(x) \text{ for } x \text{ in input()}\text{.split()}]

\# Python2: map(int, raw_input().split())

julia = scores[0]

scores = \text{sorted}([\text{julia} - x \text{ for } x \text{ in } \text{scores}[1:]]) \quad \# \text{ speichere abstand}

\[
\begin{array}{cccccccc}
J & A & B & C & D & E & F \\
5 & 4 & 1 & 1 & 2 & 3 & 1 \\
\end{array}
\]
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand
```

<table>
<thead>
<tr>
<th>5</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>F</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>A</td>
</tr>
</tbody>
</table>

```
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

print(it - 1)
```

---

J  F  B  C  D  E  A
5  1  1  1  2  3  4
Naiver Algorithmus

```
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1

print(it - 1)
```
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())

julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1

closest = 1  # wie viele sind am naehsten
while (closest < n - 1 and scores[closest] == scores[0]):
    closest += 1

print(it - 1)
```

Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1

   closest = 1  # wie viele sind am naehsten
   while (closest < n - 1 and scores[closest] == scores[0]):
       closest += 1

cutoff = int(closest / 2)  # so viele direkte verfolger kommen naeher

print(it - 1)
```

Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())

julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1

closest = 1  # wie viele sind am naehsten
    while (closest < n - 1 and scores[closest] == scores[0]):
        closest += 1

cutoff = int(closest / 2)  # so viele direkte verfolger kommen naeher

scores[0:cutoff] = [x - 1 for x in scores[0:cutoff]]  # scores[0:cutoff] = map(lambda x: x - 1, scores[0:cutoff])
scores[closest:] = [x - 1 for x in scores[closest:]]  # scores[closest:] = map(lambda x: x - 1, scores[closest:])

print(it - 1)
```

print ( i t − 1)
Naiver Algorithmus

```python
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1

closest = 1  # wie viele sind am naehesten
while (closest < n - 1 and scores[closest] == scores[0]):
    closest += 1

cutoff = int(closest / 2)  # so viele direkte verfolger kommen naeher

scores[0:cutoff] = [x - 1 for x in scores[0:cutoff]]
#scores[0:cutoff] = map(lambda x: x - 1, scores[0:cutoff])
scores[closest:] = [x - 1 for x in scores[closest:]]
#scores[closest:] = map(lambda x: x - 1, scores[closest:])

print(it - 1)
```
Python: Listen modifizieren

![Graph showing comparison of list comprehension, numpy, and map execution times against list length. The graph indicates that list comprehension is generally faster than numpy, and map is the fastest method. The execution time increases with the length of the list.](image-url)
Naiver Algorithmus: Laufzeit

```python
n = int(input())  # Python2: int(raw_input())
scores = list(map(int, input().split()))  # Python2: map(int, raw_input().split())

julia = scores[0]
scores = sorted([julia - x for x in scores[1:]]) # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1

    closest = 1  # wie viele sind am naehsten
    while (closest < n - 1 and scores[closest] == scores[0]):
        closest += 1

cutoff = int(closest / 2)  # so viele direkte verfolger kommen naeher

scores[0:cutoff] = [x - 1 for x in scores[0:cutoff]]
#scores[0:cutoff] = map(lambda x: x - 1, scores[0:cutoff])
scores[closest:] = [x - 1 for x in scores[closest:]]
#scores[closest:] = map(lambda x: x - 1, scores[closest:])

print(it - 1)
```

Naiver Algorithmus: Laufzeit

\[
n = \text{int}(\text{input}()) \quad \# \text{Python2: int(raw_input())}
\]
\[
\text{scores} = [\text{int}(x) \text{ for } x \text{ in } \text{input}().\text{split}()]
\quad \# \text{Python2: map(int, raw_input().split())}
\]
\[
\text{julia} = \text{scores}[0]
\]
\[
\text{scores} = \text{sorted}([\text{julia} - x \text{ for } x \text{ in } \text{scores}[1:]]) \quad \# \text{speichere abstand}
\]

\[
\text{it} = 0 \quad \# \text{wie viele schritte bis julia ueberholt wird}
\]

\[
\text{while scores}[0] >= 0:
\text{it} += 1
\]

\[
\text{closest} = 1 \quad \# \text{wie viele sind am naehsten}
\text{while (closest < n - 1 and scores[closest] == scores[0])}:
\quad \text{closest} += 1
\]

\[
\text{cutoff} = \text{int}(\text{closest} / 2) \quad \# \text{so viele direkte verfolger kommen naeher}
\]

\[
\text{scores}[0:\text{cutoff}] = [x - 1 \text{ for } x \text{ in } \text{scores}[0:\text{cutoff}]]
\quad \#\text{scores}[0:\text{cutoff}] = \text{map(lambda x: x - 1, scores[0:\text{cutoff}])}
\]
\[
\text{scores}[\text{closest}:] = [x - 1 \text{ for } x \text{ in } \text{scores}[\text{closest}:]]
\quad \#\text{scores}[\text{closest}:] = \text{map(lambda x: x - 1, scores[\text{closest}:])}
\]

\[
\text{print } (\text{it} - 1)
\]
Naiver Algorithmus: Laufzeit

```
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]
# Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird

while scores[0] >= 0:
    it += 1
    closest = 1  # wie viele sind am naehsten
    while (closest < n - 1 and scores[closest] == scores[0]):
        closest += 1

cutoff = int(closest / 2)  # so viele direkte verfolger kommen nahe

scores[0:cutoff] = [x - 1 for x in scores[0:cutoff]]
#scores[0:cutoff] = map(lambda x: x - 1, scores[0:cutoff])
scores[closest:] = [x - 1 for x in scores[closest:]]
#scores[closest:] = map(lambda x: x - 1, scores[closest:])

print(it - 1)
```

\( n \leq 10^5 \)

\( r \leq 10^{16} \)
Naiver Algorithmus: Laufzeit

```
n = int(input())  # Python2: int(raw_input())
scores = [int(x) for x in input().split()]  # Python2: map(int, raw_input().split())
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])  # speichere abstand

it = 0  # wie viele schritte bis julia ueberholt wird
while scores[0] >= 0:
    it += 1

closest = 1  # wie viele sind am naehesten
while (closest < n - 1 and scores[closest] == scores[0]):
    closest += 1

cutoff = int(closest / 2)  # so viele direkte verfolger kommen naeher

scores[0:cutoff] = [x - 1 for x in scores[0:cutoff]]  #scores[0:cutoff] = map(lambda x: x − 1, scores[0:cutoff])
scores[closest:] = [x - 1 for x in scores[closest:]]  #scores[closest:] = map(lambda x: x − 1, scores[closest:])

print(it - 1)
```

\[ n \cdot r \leq 10^{21} \]
Schneller?

Julia

Meisten Punkte

Weniger Punkte
Schneller?

Julia

Meisten Punkte

1 weniger

Noch weniger
Schneller?

Julia

Meisten Punkte

Distanz $k$

1 weniger $k + 1$

Noch weniger $> k + 1$
Schneller?

Julia

Meisten Punkte

Distanz

\[ k \]

1 weniger

\[ k + 1 \]

Noch weniger

\[ > k + 1 \]
Schneller?

Meisten Punkte
Distanz

1 weniger

k

k + 1

Noch weniger

> k + 1
Schneller?

Meisten Punkte

Distanz

\[ \begin{align*}
   & k \\
   & k - 1
\end{align*} \]

1 weniger

\[ \begin{align*}
   & k + 1 \\
   & k
\end{align*} \]

Noch weniger

\[ \begin{align*}
   & > k + 1 \\
   & > k
\end{align*} \]
Schneller?

Meisten Punkte
Distanz
\[ k \]
\[ k - 1 \]

1 weniger
\[ k + 1 \]
\[ k \]

Noch weniger
\[ > k + 1 \]
\[ > k \]
Schneller?

Meisten Punkte
Distanz
  \( k \)
  \( k - 1 \)

1 weniger
  \( k + 1 \)
  \( k \)
  \( k - 1 \)
  \( k \)
  \( > k \)
  \( > k + 1 \)

Noch weniger
Schneller?

Meisten Punkte
Distanz

1 weniger

Noch weniger

\[ k \]
\[ k - 1 \]

\[ k + 1 \]

\[ > k + 1 \]

\[ > k \]
Schneller?

Meisten Punkte
Distanz
\[ k \]
\[ k - 1 \]

1 weniger
\[ k + 1 \]
\[ k \]

Noch weniger
\[ > k + 1 \]
\[ > k \]
Schneller?

Meisten Punkte

Distanz

$k$

$k - 1$

$k$

1 weniger

$k + 1$

$k$

$k + 1$

Noch weniger

$> k + 1$

$> k$

$> k + 1$
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

it = 0
while distance >= 0:
    it += 1

print(it - 1)
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

it = 0
while distance >= 0:
    it += 1

print(it - 1)
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0 # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0 # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and 
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1

print(it - 1)
```python
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0  # anzahl teilnehmer mit meister punktzahl − 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1:  # oberer fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1

print(it - 1)
```
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0 # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1
almost_closest = 0 # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1: # oberer fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1
    else: # unterer fall
        closest += almost_closest

print(it - 1)
```python
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1:  # oberer fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1
    else:  # unterer fall
        closest += almost_closest
        # berechne almost_closest neu
        almost_closest = 0
        while closest + almost_closest < n - 1 and \
            scores[closest + almost_closest] <= distance + it + 1:
            almost_closest += 1

print(it - 1)
```

n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1:  # oberer fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1
    else:  # unterer fall
        closest += almost_closest

print(it - 1)
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and sorts[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1:  # oberer fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1
    else:  # unterer fall
        closest += almost_closest
        almost_closest = 0
        while closest + almost_closest < n - 1 and sorts[closest + almost_closest] <= distance + it + 1:
            almost_closest += 1

print(it - 1)
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0 # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0 # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1: # oberer fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1
    else: # unterer fall
        closest += almost_closest
        almost_closest = 0
        while closest + almost_closest < n - 1 and \
            scores[closest + almost_closest] <= distance + it + 1:
            almost_closest += 1

print(it - 1)
n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0 # Anzahl Teilnehmer mit Meister Punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1
almost_closest = 0 # Anzahl Teilnehmer mit Meister Punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    it += 1
    if closest > 1: # Oberer Fall
        almost_closest += closest - int(closest / 2)
        closest = int(closest / 2)
        distance -= 1
    else: # Unterer Fall
        closest += almost_closest
        # Berechne almost_closest neu
        almost_closest = 0
        while closest + almost_closest < n - 1 and \
            scores[closest + almost_closest] <= distance + it + 1:
            almost_closest += 1
print(it - 1)
Noch schneller?

Meisten Punkte
Distanz $k$

1 weniger
$k + 1$

Noch weniger
$> k + 1$
Noch schneller?

Meisten Punkte
Distanz $k$

1 weniger
$k + 1$

Noch weniger
$> k + 1$
Noch schneller?

Meisten Punkte

Distanz

$k$

1 weniger

$k + 1$

Noch weniger

$> k + 1$
Noch schneller?

Meisten Punkte

Distanz $k$

1 weniger $k + 1$

Noch weniger $> k + 1$
Noch schneller?

Meisten Punkte
Distanz $k$

1 weniger
$k + 1$

Noch weniger
$> k + 1$
Noch schneller?

Distanz $k$

Meisten Punkte $k$

1 weniger $k + 1$

Noch weniger $> k + 1$
Noch schneller?

Meisten Punkte

Distanz $k$

1 weniger $k + 1$

Noch weniger $> k + 1$
Noch schneller?

Meisten Punkte

Distanz $k$

$\lfloor \log_2 a \rfloor + 1$ Runden

Julia

Nach $b$ 1 weniger $k + 1$

Noch weniger $> k + 1$

Julia
Noch schneller?

Distanz $k$  

Meisten Punkte $a$  

$\lfloor \log_2 a \rfloor + 1$ Runden  

Nach $1$ weniger $a + b$  

$> k + 1$  

Noch weniger $b$  

Julia
Noch schneller?

Meisten Punkte

Distanz $k$

$\lfloor \log_2 a \rfloor + 1$ Runden

Nach $\lfloor \log_2 a \rfloor + 1$ Runden

Noch weniger

1 weniger

Julia

$1 + b \rightarrow k + 1$

$\rightarrow k + 1$

Noch weniger

$> k + 1$
Noch schneller?

Meisten Punkte

Distanz

$k$

$\lfloor \log_2 a \rfloor + 1$ Runden

Nach $\lfloor \log_2 a \rfloor + 1$ Runden

$\lfloor \log_2 a \rfloor + 1$

$a 

\rightarrow

Meisten Punkte

$a + b

b 

\rightarrow

1 weniger

$k + 1$

$\lfloor \log_2 a \rfloor$

Noch weniger

$\lfloor \log_2 a \rfloor - 1$

$\lfloor \log_2 a \rfloor$
```python
from math import log2

n = int(input())
scores = [int(x) for x in input().split()]
jug = scores[0]
scores = sorted([jug - x for x in scores[1:]])
distance = scores[0]
closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1
almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and 
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:

print (it - 1)
```
from math import log2

n = int(input())
scores = [int(x) for x in input().split()]
 julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]
closest = 0 # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1
almost_closest = 0 # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    closest_log = int(log2(closest))

print(it - 1)
from math import log2

n = int(input())
scores = [int(x) for x in input().split()]
Julia = scores[0]
scores = sorted([Julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    closest_log = int(log2(closest))
    if distance < closest_log:
        it += distance + 1
        break

print(it - 1)
from math import log2

n = int(input())
scores = [int(x) for x in input().split()]

julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0 # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0 # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and 
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    closest_log = int(log2(closest))
    if distance < closest_log:
        it += distance + 1
        break
    distance -= closest_log
    closest += almost_closest
    it += closest_log + 1

print(it - 1)
from math import log2

n = int(input())
scores = [int(x) for x in input().split()]

julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]

closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1

almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and 
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    closest_log = int(log2(closest))
    if distance < closest_log:
        it += distance + 1
        break
    distance -= closest_log
    closest += almost_closest
    it += closest_log + 1
    # finde neue almost_closest
    almost_closest = 0
    while closest + almost_closest < len(scores) and 
        scores[closest + almost_closest] <= distance + it + 1:
        almost_closest += 1

print(it - 1)
Immer noch zu langsam?
Immer noch zu langsam?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>10^{16}</td>
</tr>
<tr>
<td>A</td>
<td>10^{16}</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ a \]
Immer noch zu langsam?

\[ \begin{array}{c|c|c|c}
\text{J} & 10^{16} \\
\hline
\text{A} & 10^{16} \\
\hline
\text{B} & \text{2} \\
\hline
\text{C} & \text{1} \\
\end{array} \]

\[ \Rightarrow a \Rightarrow \text{immer noch } 10^{16} \text{ Schritte} \]
Immer noch zu langsam?

<table>
<thead>
<tr>
<th>J</th>
<th>$10^{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$10^{16}$</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

Lösung:

$\Rightarrow$ immer noch $10^{16}$ Schritte
Immer noch zu langsam?

<table>
<thead>
<tr>
<th></th>
<th>(10^{16})</th>
<th>(10^{16})</th>
<th>(2)</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lösung: Bis A und B mergen sind alle Runden gleich

⇒ immer noch \(10^{16}\) Schritte
Immer noch zu langsam?

<table>
<thead>
<tr>
<th>J</th>
<th>$10^{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$10^{16}$</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

⇒ immer noch $10^{16}$ Schritte

Lösung: Bis A und B mergen sind alle Runden gleich
⇒ Mache $\text{score}(A) - \text{score}(B)$ “Superrunden” auf einmal
Immer noch zu langsam?

<table>
<thead>
<tr>
<th>J</th>
<th>10^{16}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10^{16}</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

⇒ immer noch 10^{16} Schritte

Lösung: Bis A und B mergen sind alle Runden gleich
⇒ Mache score(A) − score(B) “Superrunden” auf einmal
⇒ (score(A) − score(B)) · (\lfloor \log_2 a \rfloor + 1) Runden
Immer noch zu langsam?

Lösung: Bis A und B mergen sind alle Runden gleich
⇒ Mache \( \text{score}(A) - \text{score}(B) \) "Superrunden" auf einmal
⇒ \((\text{score}(A) - \text{score}(B)) \cdot (\lfloor \log_2 a \rfloor + 1)\) Runden

<table>
<thead>
<tr>
<th>J</th>
<th>(10^{16})</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(10^{16})</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

\( \Rightarrow \) immer noch \(10^{16}\) Schritte

<table>
<thead>
<tr>
<th>J</th>
<th>(10^{16})</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(10^{16})</td>
</tr>
<tr>
<td>B</td>
<td>(10^{16})</td>
</tr>
<tr>
<td>C</td>
<td>(10^{16} - 1)</td>
</tr>
</tbody>
</table>
Immer noch zu langsam?

Lösung: Bis A und B mergen sind alle Runden gleich
⇒ Mache $\text{score}(A) - \text{score}(B)$ “Superrunden” auf einmal
⇒ $(\text{score}(A) - \text{score}(B)) \cdot (\lfloor \log_2 a \rfloor + 1)$ Runden

Wird in einer dieser Superrunden Julia überholt?
Immer noch zu langsam?

Lösung: Bis A und B mergen sind alle Runden gleich
⇒ Mache $\text{score}(A) - \text{score}(B)$ "Superrunden" auf einmal
⇒ $(\text{score}(A) - \text{score}(B)) \cdot (\lfloor \log_2 a \rfloor + 1)$ Runden

• Wird in einer dieser Superrunden Julia überholt?
• Was, wenn alle Konkurrenten den gleichen Score haben?
from math import log2

n = int(input())
scores = [int(x) for x in input().split()]
julia = scores[0]
scores = sorted([julia - x for x in scores[1:]])
distance = scores[0]
closest = 0  # anzahl teilnehmer mit meister punktzahl
while closest < n - 1 and scores[closest] == distance:
    closest += 1
almost_closest = 0  # anzahl teilnehmer mit meister punktzahl - 1
while closest + almost_closest < n - 1 and \
    scores[closest + almost_closest] <= distance + 1:
    almost_closest += 1

it = 0
while distance >= 0:
    closest_log = int(log2(closest))
    if distance < closest_log:
        it += distance + 1
        break
    distance -= closest_log
    closest += almost_closest
    it += closest_log + 1
    # finde neue almost_closest
    almost_closest = 0
    while closest + almost_closest < len(scores) and \
        scores[closest + almost_closest] <= distance + it + 1:
        almost_closest += 1
print(it - 1)