[redacted]

Question - 1
Concurrent Access Limit Breach

Find all unique user IDs who breached the limit by having more than *m* concurrent sessions at any point. Each "login" increases the user's session count by 1, and each "logout" decreases it by 1 only if the count is above 0.

The function *getLimitBreaches* will take four inputs:
int timestamps[n]: a non-decreasing array of event times
string userIds[n]: user ID for each event
string actions[n]: action for each event, either "login" or "logout."
int m: the maximum allowed number of concurrent sessions per user

The function should return a lexicographically sorted array of user IDs who breached the limit. If no user breaches the limit, return an empty array.

Example
timestamps = [1, 2, 3, 4]
userIds = ["A", "A", "A", "A"]
actions = ["login", "login", "login", "logout"]
m = 2

The table below shows the session timeline for each event, underlining the point at which the user's active session count exceeded the allowed limit:

User Session Timeline

Time	User ID	Number of Active Sessions
1	"A"	1
2	"A"	2
<u>3</u>	<u>"A"</u>	<u>3</u>
4	"A"	2

At timestamp 3, user "A" exceeds the session limit *m* = 2 with 3 consecutive logins.
Hence, the answer is ["A"].

Constraints
- $1 \le n \le 2 * 10^5$
- $0 \le timestamps[i] \le 10^9$
- 1 ≤ length of *userIds[i]* ≤ 10
- It is guaranteed that *userIds[i]* consists of uppercase alphanumeric characters.
- *actions[i]* is in the set {"login", "logout"}
- $0 \le m \le 10^5$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the number of log entries.

The next *n* lines each contain an integer *timestamps[i]*.
The next line contains an integer *n*, the size of the *userIds* array.
The next *n* lines each contain a string *userIds[i]*.
The next line contains an integer *n*, the size of the *actions* array.
The next *n* lines each contain a string *actions[i]*.
The final line contains an integer *m*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
6       →     timestamps[] size n = 6
10      →     timestamps = [10, 20, 30, 40, 50, 60]
20
30
40
50
60
6       →     userIds[] size n = 6
A       →     userIds = ["A", "B", "A", "A", "B", "A"]
B
A
A
B
A
6       →     actions[] size n = 6
login   →     actions = ["login", "login", "login", "login", "logout", "login"]
login
login
login
logout
login
2       →     m = 2
```

Sample Output 0

```
A
```

Explanation

User Session Timeline

Time	User ID	Number of Active Sessions
10	"A"	1
20	"B"	1
30	"A"	2
40	"A"	3
50	"B"	0
60	"A"	4

User "A" performs multiple logins without timely logouts, reaching 3 sessions at timestamp 40 and 4 at 60, breaching the limit *m* = 2. User "B" stays within the limit.
Hence, the answer is ["A"].

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
6       →     timestamps[] size n = 6
5       →     timestamps = [5, 10, 15, 20, 25, 30]
10
15
20
25
30
6       →     userIds[] size n = 6
A       →     userIds = ["A", "B", "A", "B", "B", "A"]
B
A
B
B
A
6       →     actions[] size n = 6
login   →     actions = ["login", "login", "login", "login", "logout", "logout"]
login
login
login
logout
logout
2       →     m = 2
```

Sample Output 1

```

```

Explanation

User Session Timeline

Time	User ID	Number of Active Session
5	"A"	1
10	"B"	1
15	"A"	2
20	"B"	2
25	"B"	0
30	"A"	0

Users "A" and "B" each perform two logins and one logout without exceeding the session limit *m* = 2 at any point.
Hence, the answer is [].

Question - 2
Server Downtime Counter

Given the monitoring period length *t* and arrays *startTime* and *endTime* representing the inclusive up-time intervals, find the total downtime of the server within the interval [1, *t*].

The function *getTotalDowntime* will take three inputs:
int startTime[n]: starting time (in seconds) of each up-time interval
int endTime[n]: ending time (in seconds) of each up-time interval
int t: the total monitoring time (in seconds)

The function should return an integer representing the total downtime (in seconds) of the server within [1, *t*].

Example

n = 2

startTime = [1, 2]

endTime = [3, 5]

t = 10

The server was up from [1, 3] and [2, 5], which overlap to cover [1, 5], so the downtime interval is [6, 10].
Hence, the answer is 5.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le t \le 10^9$
- $1 \le startTime[i] \le endTime[i] \le t$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *startTime*.
Each of the next *n* lines contains an integer *startTime[i]*.
The next line contains an integer *n*, the size of the array *endTime*.
Each of the next *n* lines contains an integer *endTime[i]*.
The next line contains an integer *t*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
3         →    startTime[] size n = 3
1         →    startTime = [1, 2, 7]
2
7
3         →    endTime[] size n = 3
2         →    endTime = [2, 4, 8]
4
8
9         →    t = 9
```

Sample Output 0

```
3
```

Explanation

The server was up from [1, 2], [2, 4], and [7, 8], which overlap to cover [1, 4] and [7, 8], so downtime intervals are [5, 6] and [9, 9].
Hence, the answer is 3.

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
2         →    startTime[] size n = 2
1         →    startTime = [1, 2]
2
2         →    endTime[] size n = 2
2         →    endTime = [2, 9]
9
9         →    t = 9
```

Sample Output 1

```
0
```

Explanation

The server was up from [1, 2] and [2, 9], which overlap to cover [1, 9], so there is no downtime interval.

Question - 3
Max Retry Within Window

Given a series of request timestamps, find the maximum number of requests that occur within any continuous time window of a specified length.

The function *maxRequestInWindow* will take two inputs:
int timestamp[n]: request timestamps in minutes
int windowSize: duration of the time window in minutes

The function should return an integer denoting the maximum requests observed in any window of *windowSize* minutes.

Example
timestamp = [1, 3, 7, 5]
windowSize = 4

Some windows are:
- [1, 4]: includes time 1 and 3 → total requests = 2.
- [3, 6]: includes time 3 and 5 → total requests = 2.
- [5, 8]: includes time 5 and 7 → total requests = 2.

It can be seen that the maximum number of requests within the window size is 2.
Hence, the answer is 2.

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le timestamp[i] \le 10^9$
- $1 \le windowSize \le 10^9$

▼ Input Format for Custom Testing

The first line contains the integer *n,* the size of the array *timestamp*.
The next *n* lines contain an integer *timestamp[i]*.
The next line contains an integer *windowSize*.

▼ Sample Case 0

Sample Input 0

```
STDIN            Function
-----            --------
3         →      timestamp[] size n = 3
2         →      timestamp = [2, 2, 3]
2
3
1         →      windowSize = 1
```

Sample Output 0

```
2
```

Explanation
- [2, 2]: includes both requests at time 2 → total requests = 2.
- [3, 3]: includes only time 3 → total requests = 1.

▼ Sample Case 1

Sample Input 1

```
STDIN               Function
-----               --------
3          →        timestamp[] size n = 3
1          →        timestamp = [1, 2, 3]
2
3
4          →        windowSize = 4
```

Sample Output 1

```
3
```

Explanation
All the requests can be included in a single window, so the answer is 3.

Question - 4
Top K APIs by Load

Given a list of events, each with associated API indices and corresponding loads, find the API indices of the top *k* APIs with the highest total cumulative load across all events. If multiple APIs have the same total cumulative load, choose the one with the smaller index.

The function *getTopKAPIs* will take three inputs:
int apiIndex[n]: indices of APIs called
int loads[n]: loads generated by each corresponding API call
int k: number of top API indices to return

The function should return an array of *k* integers representing the API indices, sorted in descending order of total cumulative load. If multiple APIs have the same total cumulative load, APIs with smaller indices appear first.

Example
n = 3
apiIndex = [1, 2, 3]
loads = [6, 10, 15]
k = 2

The two highest loads are 15 and 10 with *apiIndex* values of 3 and 2, respectively.
Hence, the answer is [3, 2].

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le apiIndex[i] \le n$
- $1 \le loads[i] \le 10^8$
- $1 \le k \le n$
- It is guaranteed that there are at least *k* distinct API IDs with non-zero total load values.

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *apiIndex*.
Each of the next *n* lines contains an integer *apiIndex[i]*.
The next line contains an integer *n*, the size of the array *loads*.
Each of the next *n* lines contains an integer *loads[i]*.
The next line contains an integer k.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
3         →    apiIndex[] size n = 3
1         →    apiIndex = [1, 2, 1]
2
1
3         →    loads[] size n = 3
2         →    loads = [2, 5, 3]
5
3
1         →    k = 1
```

Sample Output 0

```
1
```

Explanation

API indices 1 and 2 have the same cumulative load of 5, but API index 1 has a smaller index.
Hence, the answer is [1].

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
3         →    apiIndex[] size n = 3
1         →    apiIndex = [1, 2, 1]
2
1
3         →    loads[] size n = 3
2         →    loads = [2, 4, 3]
4
3
2         →    k = 2
```

Sample Output 1

```
1
2
```

Explanation

The API index 1 has the highest load, which is 2 + 3 = 5, and the API index 2 has a load of 4.
Hence, the answer is [1, 2].

Question - 5
API Rate Limiter Implementation

You are building a rate limiter for an API service. Given a list of requests, where each request has a user ID and a timestamp, determine if the request should be allowed.

A request is allowed only if the user has made fewer than *k* requests in the past 1-minute sliding window.

The function *getAllowedRequests* will take three inputs:
int user[n]: the IDs of the users
int timestamp[n]: the timestamp of requests in seconds
int k: the allowed number of requests in any 1-minute sliding window

The function should return an array of integers where the i^{th} element is 1 if the i^{th} request is allowed; otherwise, it should be 0.

Note: If a request is not allowed due to the rate limit, it is not counted as a valid request for future rate limit counting.

Example
n = 3
user = [1, 1, 1]
timestamp = [6, 10, 65]
k = 2

Request Evaluation per User

Request Index (0-based)	User ID	Timestamp	User's Requests in Last 60s (Allowed)	Request Count if current request is processed	Allowed
0	1	6	[]	1	Yes
1	1	10	[6]	2	Yes
2	1	65	[6, 10]	3	No

Hence, the answer is [1, 1, 0]. The third request would have been allowed if it had arrived at timestamp 66.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ *user[i]* ≤ *n*
- 1 ≤ *timestamp[i]* ≤ 10^9
- 1 ≤ *k* ≤ *n*
- It is guaranteed that the array *timestamp* is sorted in non-decreasing order.

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *user*.
Each of the next *n* lines contains an integer *user[i]*.
The first line contains an integer *n*, the size of the array *timestamp*.
Each of the next *n* lines contains an integer *timestamp[i]*.
The first line contains an integer k.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
3         →    user[] size n = 3
1         →    user = [1, 2, 3]
2
3
3         →    timestamp[] size n = 3
1         →    timestamp = [1, 10, 70]
10
70
1         →    k = 1
```

Sample Output 0

```
1
1
1
```

Explanation

Request Evaluation per User

Request Index (0-based)	User ID	Timestamp	User's Requests in Last 60s (Allowed)	Request Count if current request is processed	Allowed
0	1	1	[]	1	Yes
1	2	10	[]	1	Yes
2	3	70	[]	1	Yes

Hence, the answer is [1, 1, 1].

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
3         →    user[] size n = 3
1         →    user = [1, 1, 1]
1
1
3         →    timestamp[] size n = 3
1         →    timestamp = [1, 10, 20]
10
20
1         →    k = 1
```

Sample Output 1

```
1
0
0
```

Explanation

Request Evaluation per User

Request Index (0-based)	User ID	Timestamp	User's Requests in Last 60s (Allowed)	Request Count if current request is processed	Allowed
0	1	1	[]	1	Yes
1	1	10	[1]	2	No
2	1	20	[1]	2	No

Hence, the answer is [1, 0, 0].

Question - 6
Detect IP Access Pattern

Given arrays of IP addresses with their respective timestamps, find all IP addresses that appear more than *k* times within any *t*-second window. If no such IP addresses exist, return an empty array.

The function *getSpammerIPs* will take four inputs:
  int ip[n]: array of IP addresses for each access
  int timestamp[n]: array of timestamps
  int k: threshold for number of accesses (must be exceeded)
  int t: the size of the time window in seconds

The function should return a sorted array of IP addresses that occur in more than *k* accesses within any *t*-second window. If no such IP exists, return an empty array.

Note: Two timestamps *x* and *y* are in the same *t*-second window if the absolute difference *y* - *x* < *t*, assuming timestamps are in ascending order for a given IP.

Example

n = 3

ip = [1, 2, 1]

timestamp = [6, 10, 15]

k = 1

t = 10

Access Check

IP	Access Timestamps (seconds)	Maximum count within any window of *t* = 10 seconds
1	[6, 15]	2 accesses in window [6, 15]
2	[10]	Only 1 access

Hence, the answer is [1].

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le ip[i] \le n$
- $1 \le timestamp[i], t \le 10^9$
- $1 \le k \le n$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *ip*.

Each of the next *n* lines contains an integer *ip[i]*.

The next line contains an integer *n*, the size of the array *timestamp*.

Each of the next *n* lines contains an integer *timestamp[i]*.

The next line contains an integer *k*.

The next line contains an integer *t*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
4         →    ip[] size n = 4
1         →    ip = [1, 1, 2, 2]
1
2
2
4         →    timestamp[] size n = 4
1         →    timestamp = [1, 2, 3, 4]
2
3
4
1         →    k = 1
5         →    t = 5
```

Sample Output 0

```
1
2
```

Explanation

Access Check

IP	Access Timestamps (seconds)	Maximum count within any window of *t* = 5 seconds
1	[1, 2]	2 accesses in window [1, 5]
2	[3, 4]	2 accesses in window [3, 7]

Hence, the answer is [1, 2].

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
4         →    ip[] size n = 4
1         →    ip = [1, 1, 2, 2]
1
2
2
4         →    timestamp[] size n = 4
1         →    timestamp = [1, 2, 3, 4]
2
3
4
2         →    k = 2
5         →    t = 5
```

Sample Output 1

```
```

Explanation

Access Check

IP	Access Timestamps (seconds)	Maximum count within any window of *t* = 5 seconds
1	[1, 2]	2 accesses in window [1, 5]
2	[3, 4]	2 accesses in window [3, 7]

Hence, the answer is [].

Question - 7
API Rate Limiter

You are building a rate limiter that controls how many requests a user can make over time. Given a list of requests, where each request has a user ID and a timestamp, determine if the request should be allowed.

A request is allowed only if the user has made fewer than *k* requests in the past *t*-second sliding window.

The function *getRequests* will take four inputs:

int user[n]: the ID of the user making the i^{th} request
int time[n]: the timestamp (in seconds) of that request
int k: the maximum number of requests each user can make within any *t*-seconds
int t: the window size

The function should return an array of integers, with the i^{th} element being 1 if the i^{th} request is allowed; otherwise, it should be 0.

Note: If a request is not allowed, it is not counted for future rate limit counting.

Example

n = 3

user = [1, 1, 1]

time = [6, 10, 15]

k = 2

t = 10

Request Evaluation per User

Request Index (0-based)	User ID	Timestamp	User's Requests in Last 10s (Allowed)	Request Count	Allowed
0	1	6	[6]	1	Yes
1	1	10	[6, 10]	2	Yes
2	1	15	[6, 10, 15]	3	No

Hence, the answer is [1, 1, 0].

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le user[i] \le n$
- $1 \le time[i] \le 10^9$
- $1 \le k \le n$
- $1 \le t \le 10^9$
- It is guaranteed that the array *time* is sorted in non-decreasing order.

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *user*.

Each of the next *n* lines contains an integer *user[i]*.

The next line contains an integer *n*, the size of the array *time*.

Each of the next *n* lines contains an integer *time[i]*.

The next line contains an integer *k*.

The next line contains an integer *t*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
2         →    user[] size n = 2
1         →    user = [1, 2]
2
2         →    time[] size n = 2
1         →    time = [1, 10]
10
1         →    k = 1
1         →    t = 1
```

Sample Output 0

```
1
1
```

Explanation

Request Evaluation per User

Request Index (0-based)	User ID	Timestamp	User’s Requests in Last 1s (Allowed)	Request Count	Allowed
0	1	1	[1]	1	Yes
1	2	10	[10]	1	Yes

Hence, the answer is [1, 1].

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
2          →   user[] size n = 2
1          →   user = [1, 1]
1
2          →   time[] size n = 2
1          →   time = [1, 2]
2
1          →   k = 1
2          →   t = 2
```

Sample Output 1

```
1
0
```

Explanation

Request Evaluation per User

Request Index (0-based)	User ID	Timestamp	User’s Requests in Last 2s (Allowed)	Request Count	Allowed
0	1	1	[1]	1	Yes
1	1	2	[1, 2]	2	No

Hence, the answer is [1, 0].

Question - 8
Anomaly Spike Index

You are given a list of numbers representing recorded measurements. A value is called a spike if it is at least three times greater than the maximum of all previous values.

Find the 0-based index of the first spike in the stream. If no spike is found, return -1.

The function *detectFirstAnomaly* will take one input:
int metrics[n]: the monitored data stream values

The function should return an integer denoting the 0-based index of the first spike, or -1 if no such spike is found.

Example
n = 4
metrics = [3, 10, 2, 7]

The analysis of metric values is as follows (using 0-based indexing):

- At index 0: The value is 3. Being the first data point, it becomes the initial maximum observed.
- At index 1: The value is 10. The current maximum is 3, and since 10 ≥ 3 * 3 = 9, this qualifies as a spike.

As a spike is detected at index 1, further values are not evaluated.

Hence, the answer is 1.

Constraints

- $2 \le n \le 2 * 10^5$
- $1 \le metrics[i] \le 10^9$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *metrics*.
Each of the next *n* lines (where 0 ≤ *i* < *n*) contains an integer, *metrics[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN               Function
-----               --------
3          →        metrics[] size n = 3
2          →        metrics = [2, 1, 5]
1
5
```

Sample Output 0

```
-1
```

Explanation

- At index 0: The value is 2. Being the first data point, it becomes the initial maximum observed.
- At index 1: The value is 1. The current maximum is 2, and since 1 < 3 * 2 = 6, this is not a spike.
- At index 2: The value is 5. The current maximum is still 2, and since 5 < 3 * 2 = 6, this is also not a spike.

There are no spikes.

▾ Sample Case 1

Sample Input 1

```
STDIN               Function
-----               --------
5          →        metrics[] size n = 5
1          →        metrics = [1, 2, 5, 15, 45]
2
5
15
45
```

Sample Output 1

```
3
```

Explanation

- At index 0: The value is 1. Being the first data point, it becomes the initial maximum observed.
- At index 1: The value is 2. The current maximum is 1, and since 2 < 3 * 1 = 3, this is not a spike. Update the maximum to 2.
- At index 2: The value is 5. The current maximum is 2, and since 5 < 3 * 2 = 6, this is not a spike. Update the maximum to 5.
- At index 3: The value is 15. The current maximum is 5, and since 15 ≥ 3 * 5 = 15, this is a spike.

A spike is detected at index 3.

Question - 9

System Load Monitor

Identify time periods when a system experiences high load based on a rolling average calculation. For each minute in the monitoring period, calculate the average load over the most recent window of minutes. If this average exceeds the specified threshold, include that minute in your results.

The function *detectHighLoadWindows* will take three inputs:
int loads[n]: system load at each minute *i* (0-based)
int windowSize: size of the rolling window in minutes
int threshold: average load threshold to compare against

The function returns a sorted array of minute indices where high load was detected, or an empty array if no such minutes exist.

Example
n = 4
loads = [8, 14, 5, 17]
windowSize = 2
threshold = 10

All possible windows of size 2

Window Range	Average Load	Is the average greater than *threshold*?
[0, 1]	(8 + 14) / 2 = 11	Yes
[1, 2]	(14 + 5) / 2 = 9.5	No
[2, 3]	(5 + 17) / 2 = 11	Yes

At minute 0, only one value is available, which is less than the window size, so it is skipped.
Hence, the answer is [1, 3].

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le windowSize \le n$
- $1 \le loads[i], threshold \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *loads*.
The next *n* lines contain an integer, *loads[i]*.
The next line contains an integer, *windowSize*.
The last line contains an integer, *threshold*.

▼ Sample Case 0

Sample Input 0

```
STDIN              Function
-----              --------
3          →       loads[] size n = 3
5          →       loads = [5, 1, 3]
1
3
2          →       windowSize = 2
2          →       threshold = 2
```

Sample Output 0

```
1
```

Explanation

All possible windows of size 2

Window Range	Average Load	Is the average greater than *threshold*?
[0, 1]	(5 + 1) / 2 = 3	Yes
[1, 2]	(1 + 3) / 2 = 2	No

Hence, the answer is [1].

▾ Sample Case 1

Sample Input 1

```
STDIN              Function
-----              --------
5          →       loads[] size n = 5
1          →       loads = [1, 10, 7, 1, 10]
10
7
1
10
3          →       windowSize = 3
5          →       threshold = 5
```

Sample Output 1

```
2
3
4
```

Explanation

All possible windows of size 3

Window Range	Average Load	Is the average greater than *threshold*?
[0, 2]	(1 + 10 + 7) / 3 = 6	Yes
[1, 3]	(10 + 7 + 1) / 3 = 6	Yes
[2, 4]	(7 + 1 + 10) / 3 = 6	Yes

Hence, the answer is [2, 3, 4].

Question - 10
Server Load Snapshot

Find the maximum total load observed within any window of a given size among a series of load measurements at specific timestamps.

The function *maxLoadInWindow* takes three inputs:
  int timestamp[n]: time stamps in minutes
  int load[n]: load at each timestamp

int windowSize: window size in minutes

The function should return a long integer denoting the maximum total load observed in any window of *windowSize* minutes.

Example
timestamp = [1, 3, 7, 5]
load = [10, 20, 40, 30]
windowSize = 4

Some of the windows are:
- [1, 4]: includes time 1 and 3 → load = 10 + 20 = 30
- [3, 6]: includes time 3 and 5 → load = 20 + 30 = 50
- [5, 8]: includes time 5 and 7 → load = 30 + 40 = 70

The maximum load within the window size is 70, so the answer is 70.

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le$ *timestamp[i], load[i]* $\le 10^9$
- $1 \le$ *windowSize* $\le 10^9$

▼ Input Format for Custom Testing

The first line contains the integer *n,* the size of the array *timestamp*.
The next *n* lines contain an integer *timestamp[i]*.
The next line contains the integer *n,* the size of the array *load*.
The next *n* lines contain an integer *load[i]*.
The next line contains an integer *windowSize*.

▼ Sample Case 0

Sample Input 0

```
STDIN           Function
-----           --------
3         →     timestamp[] size n = 3
2         →     timestamp = [2, 2, 3]
2
3
3         →     load[] size n = 3
5         →     load = [5, 7, 10]
7
10
1         →     windowSize = 1
```

Sample Output 0

```
12
```

Explanation
Some windows are:

- [2, 2]: includes both entries at time 2 → load = 5 + 7 = 12
- [3, 3]: includes only time 3 → load = 10

▼ Sample Case 1

Sample Input 1

```
STDIN           Function
-----           --------
3         →     timestamp[] size n = 3
1         →     timestamp = [1, 2, 3]
2
3
3         →     load[] size n = 3
```

```
4          →      load = [4, 5, 6]
5
6
5          →      windowSize = 5
```

Sample Output 1

```
15
```

Explanation
All the activities' timestamps can be included in a single *windowSize* interval.
Hence, the answer is 4 + 5 + 6 = 15.

Question - 11
Database Query Optimizer

You are optimizing database queries by merging overlapping time ranges. Given time intervals, merge all overlapping intervals to reduce redundant database calls.

The function *getMergedIntervalStarts* will take two inputs:
int startTime[n]: start times of the intervals
int endTime[n]: the corresponding end times

The function should return a sorted array of integers that denote the starting times of the merged intervals.

Example
n = 5
startTime = [1, 3, 2, 7, 15]
endTime = [3, 6, 4, 10, 18]

The following are the intervals after merging:
- [1, 3], [2, 4], and [3, 6] all overlap → merged into [1, 6].
- [7, 10] → no overlap → stays as is.
- [15, 18] → no overlap → stays as is.

Therefore, the merged intervals are: [1, 6], [7, 10], [15, 18], and the answer is [1, 7, 15].

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le$ *startTime[i], endTime[i]* $\le 10^9$
- It is guaranteed that *startTime[i]* ≤ *endTime[i]*.

▼ Input Format for Custom Testing

The first line contains the integer *n,* the size of the array *startTime*.
The next *n* lines contain an integer *startTime[i]*.
The next line contains the integer *n,* the size of the array *endTime*.
The next *n* lines contain an integer *endTime[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN             Function
-----             --------
4          →      startTime[] size n = 4
1          →      startTime = [1, 2, 3, 4]
2
3
4
4          →      endTime[] size n = 4
```

```
5         →      endTime = [5, 6, 7, 8]
6
7
8
```

Sample Output 0

```
1
```

Explanation

All intervals overlap and form one merged interval, [1, 8].

The answer is [1].

▼ Sample Case 1

Sample Input 1

```
STDIN              Function
-----              --------
3         →      startTime[] size n = 3
1         →      startTime = [1, 3, 4]
3
4
3         →      endTime[] size n = 3
2         →      endTime = [2, 3, 6]
3
6
```

Sample Output 1

```
1
3
4
```

Explanation

No intervals overlap, so no merging is needed.

The merged intervals are: [1, 2], [3, 3], [4, 6].

Question - 12
Maximum Stock Profit

A trading application receives *n* stock prices recorded throughout the day. To advise users, implement a function that determines the maximum profit obtainable by buying once and selling once. You must buy before you sell. If no profit is possible, return 0.

The function *maxProfit* will take one input:

int prices[n]: the stock price recorded throughout the day in order

The function should return an integer denoting the maximum profit achievable under the given constraints.

Example

n = 6

prices = [8, 1, 5, 3, 6, 4]

The best strategy is to buy at the 2nd minute when the price is 1 and sell at the 5th minute when the price is 6.

So, the maximum profit = 6 - 1 = 5.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le prices[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains the integer *n,* the size of the array *prices*.

The next *n* lines contain an integer *prices[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN             Function
-----             --------
5         →       prices[] size n = 5
2         →       prices = [2, 4, 6, 8, 10]
4
6
8
10
```

Sample Output 0

```
8
```

Explanation

Since the stock price increases every minute, the optimal strategy is to buy at the first minute when the price is 2 and sell at the last minute when the price is 10.

This gives the maximum profit of 10 - 2 = 8.

▼ Sample Case 1

Sample Input 1

```
STDIN               Function
-----               --------
5          →        prices[] size n = 5
10         →        prices = [10, 8, 6, 6, 4]
8
6
6
4
```

Sample Output 1

```
0
```

Explanation

In this case, the stock price decreases each minute, so there is no way to make a profit.

The best strategy is to do nothing, for 0 profit.

Question - 13

API Retry Analyzer

Given arrays of timestamps and request IDs, find the total number of retries across all request IDs. A retry is defined as two consecutive logs for the same request ID recorded with a time gap of less than or equal to *gap* seconds. The timestamps array is sorted in non-decreasing order.

The function *getRetryCount* will take three inputs:

int gap: the maximum allowed time difference (in seconds) to consider a retry

string requestIds[n]: request ID for each log

int timestamps[n]: timestamp for each log

The function should return an integer denoting the total number of retries made by all request IDs.

Example

gap = 10

requestIds = ["r1", "r1", "r1", "r2", "r2"]
timestamps = [100, 105, 200, 300, 302]

The table below shows the total number of retries for each request ID:

Request Retry Table

Request ID	Timestamps	Retry Pairs	Retry Count
r1	[100, 105, 200]	(100, 105)	1
r2	[300, 302]	(300, 302)	1

The total number of retries = 1 + 1 = 2.
Hence, the answer is 2.

Constraints

- 1 ≤ *gap* ≤ 10^9
- 1 ≤ *n* ≤ 2 * 10^5
- 0 ≤ *timestamps[i]* ≤ 10^9
- 1 ≤ *length of requestIds[i]* ≤ 10
- It is guaranteed that *requestIds[i]* is an alphanumeric string.
- It is guaranteed that the *timestamps* array is sorted in non-decreasing order.

▼ Input Format for Custom Testing

The first line contains an integer *gap*.
The second line contains an integer *n*, the number of events.
The next *n* lines each contain a string *requestIds[i]*.
The next line contains an integer *n*, the size of *timestamps*.
The next *n* lines each contain an integer *timestamps[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
15       →    gap = 15
6        →    requestIds[] size n = 6
r2       →    requestIds = ["r2", "r1", "r2", "r1", "r2", "r3"]
r1
r2
r1
r2
r3
6        →    timestamps[] size n = 6
5        →    timestamps = [5, 10, 18, 20, 40, 100]
10
18
20
40
100
```

Sample Output 0

```
2
```

Explanation

Request Retry Table

Request ID	Timestamps	Retry Pairs	Retry Count
r1	[10, 20]	(10, 20)	1
r2	[5, 18, 40]	(5, 18)	1
r3	[100]	-	0

The total number of retries = 1 + 1 + 0 = 2.

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
5       →      gap = 5
5       →      requestIds[] size n = 5
x1      →      requestIds = ["x1", "x1", "x1", "x2", "x2"]
x1
x1
x2
x2
5       →      timestamps[] size n = 5
100     →      timestamps = [100, 103, 107, 200, 208]
103
107
200
208
```

Sample Output 1

```
2
```

Explanation

Request Retry Table

Request ID	Timestamps	Retry Pairs	Retry Count
x1	[100, 103, 107]	(100, 103), (103, 107)	2
x2	[200, 208]	-	0

The total number of retries = 2 + 0 = 2.

Question - 14
Session Reconstructor

Given arrays of timestamps for user events and corresponding user IDs, find the total number of sessions across all users. A session for a user is defined as a contiguous sequence of events such that no two consecutive events for that user are separated by more than *timeout* seconds.

Any gap greater than *timeout* seconds between consecutive events for the same user starts a new session.

The function *getSessionCount* will take three inputs:

int timeout: the maximum allowed gap (in seconds) between two consecutive events for a session

string userIds[n]: user ID for each event

int timestamps[n]: timestamp (in seconds) for each event

The function should return an integer denoting the total number of sessions across all users.

Example

timeout = 10

userIds = ["u1", "u1", "u1", "u2", "u2", "u2", "u2"]

timestamps = [1, 5, 17, 3, 10, 20, 35]

The table below shows the total number of sessions for each user:

User Session Table

User	Timestamps	Sessions	Session Counts
u1	[1, 5, 17]	[1, 5], [17]	2
u2	[3, 10, 20, 35]	[3, 10, 20], [35]	2

The total number of sessions for all users = 2 + 2 = 4.

Hence, the answer is 4.

Constraints

- 1 ≤ *timeout* ≤ 10^9
- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ length of *userIds[i]* ≤ 10
- 0 ≤ *timestamps[i]* ≤ 10^9
- It is guaranteed that *userIds[i]* is an alphanumeric string.

▼ Input Format for Custom Testing

The first line contains an integer *timeout*.

The second line contains an integer *n*, the number of events.

The next *n* lines each contain a string *userIds[i]*.

The next line contains an integer *n*, the number of *timestamps*.

The next *n* lines each contain an integer *timestamps[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
10      →      timeout = 10
7       →      userIds[] size n = 7
u1      →      userIds = ["u1", "u1", "u2", "u2", "u2", "u3", "u3"]
u1
u2
u2
u2
u3
u3
7       →      timestamps[] size n = 7
1       →      timestamps = [1, 5, 2, 15, 30, 10, 25]
5
2
```

```
15
30
10
25
```

Sample Output 0

```
6
```

Explanation

User Session Table

User	Timestamps	Sessions	Session Counts
u1	[1, 5]	[1, 5]	1
u2	[2, 15, 30]	[2], [15], [30]	3
u3	[10, 25]	[10], [25]	2

The total number of sessions for all users = 1 + 3 + 2 = 6.

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
11      →     timeout = 11
6       →     userIds[] size n = 6
u1      →     userIds = ["u1", "u1", "u1", "u2", "u2", "u2"]
u1
u1
u2
u2
u2
6       →     timestamps[] size n = 6
2       →     timestamps = [2, 9, 25, 1, 12, 30]
9
25
1
12
30
```

Sample Output 1

```
4
```

Explanation

User Session Table

User	Timestamps	Sessions	Session Counts
u1	[2, 9, 25]	[2, 9], [25]	2
u2	[1, 12, 30]	[1, 12], [30]	2

The total number of sessions for all users = 2 + 2 = 4.

Question - 15
Peak API Load Window

Given an array of timestamps and a window size, find the maximum number of requests made within any time window defined as the closed interval [*x*, *x* + *window* – 1] for some integer *x*.

The function *peakAPIWindow* will take two inputs:
int window: the size of the time window
int timestamps[n]: array of request timestamps

The function should return an integer denoting the maximum number of requests made within any given window size.

Example
window = 5
timestamps = [1, 2, 3, 8, 10]

The window [0, 4] contains 3 requests at timestamps 1, 2, and 3, which is the maximum among all windows of size 5.
Hence, the answer is 3.

Constraints

- $1 \le window \le 10^9$
- $1 \le n \le 2 * 10^5$
- $0 \le timestamps[i] \le 10^9$
- It is guaranteed that *timestamps* are sorted in non-decreasing order.

▼ Input Format for Custom Testing

The first line contains an integer *window*.
The second line contains an integer *n*, the size of the array *timestamps*.
The next *n* lines each contain an integer *timestamps[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
4       →      window = 4
6       →      timestamps[] size n = 6
2       →      timestamps = [2, 3, 4, 5, 10, 12]
3
4
5
10
12
```

Sample Output 0

```
4
```

Explanation
In the time window [2, 5], there are 4 requests (timestamps 2, 3, 4, 5), which is the maximum number of requests in any window of size 4.

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
```

```
4       →     window = 4
8       →     timestamps[] size n = 8
1       →     timestamps = [1, 2, 3, 10, 11, 12, 20, 21]
2
3
10
11
12
20
21
```

Sample Output 1

```
3
```

Explanation

In the time window [0, 3], there are 3 requests (timestamps 1, 2, 3), which is the maximum number of requests in any window of size 3.

Question - 16

Frequent Query Detector

Given a list of queries, each associated with a timestamp and a query type, find all query types that appear at least *threshold* times within any 600-second window.

The function *getFrequentQueries* will take three inputs:

int threshold: the minimum number of times a query type must appear in any window for inclusion

int timestamps[n]: a non-decreasing array of query timestamps in seconds

string queryTypes[n]: an array of query types for each corresponding timestamp

The function should return a string array of query types that appear at least threshold times within any 600-second window. The returned list should be in sorted order. If there is no such query type, return an empty string array.

Example

threshold = 2

timestamps = [100, 150, 200, 250, 700]

queryTypes = ["Q1", "Q1", "Q1", "Q2", "Q1"]

The table below shows the time window during which the query type appeared at least *threshold* times, along with its corresponding frequency count:

Query Types Frequency Table

TimeWindow	QueryType	FrequencyCount

[101, 700]	Q1	3

The query type "Q1" occurs 3 times between timestamps 101 and 700, which is within a 600-second window. Since *threshold* = 2, "Q1" exceeds the required frequency.
Hence, the answer is ["Q1"].

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ *threshold* ≤ 2 * 10^5
- 1 ≤ length of *queryTypes[i]* ≤ 10
- 0 ≤ *timestamps[i]* ≤ 10^9
- It is guaranteed that *queryTypes[i]* consists of uppercase alphanumeric characters.

▼ Input Format for Custom Testing

The first line contains an integer *threshold*.
The second line contains an integer *n*, the size of the array *timestamps*.
The next *n* lines each contain an integer *timestamps[i]*.
The next line contains an integer *n*, the size of the array *queryTypes*.
The next *n* lines each contain a string *queryTypes[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
2       →      threshold = 2
7       →      timestamps[] size n = 7
100     →      timestamps = [100, 150, 200, 250, 300, 350, 700]
150
200
250
300
350
700
7       →      queryTypes[] size n = 7
Q1      →      queryTypes = ["Q1", "Q2", "Q1", "Q2", "Q2", "Q1", "Q1"]
Q2
Q1
Q2
Q2
Q1
Q1
```

Sample Output 0

```
Q1
Q2
```

Explanation

The table below shows the taken time window during which query type appeared at least *threshold* times, along with its corresponding frequency count:

Query Types Frequency Table

Time Window	Query Type	Frequency Count
[101, 700]	Q1	3

[101, 700]	Q2	3

The query types "Q1" and "Q2" each occur 3 times in the respective time window. Since *threshold* = 2, both query types meet the frequency requirement. Hence, the answer is ["Q1", "Q2"].

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
3       →      threshold = 3
5       →      timestamps[] size n = 5
101     →      timestamps = [101, 200, 250, 700, 800]
200
250
700
800
5       →      queryTypes[] size n = 5
Q1      →      queryTypes = ["Q1", "Q1", "Q2", "Q1", "Q2"]
Q1
Q2
Q1
Q2
```

Sample Output 1

```
Q1
```

Explanation

The table below shows the taken time window during which query type appeared at least *threshold* times, along with its corresponding frequency count:

Query Types Frequency Table

TimeWindow	QueryType	FrequencyCount
[101, 700]	Q1	3

The query type "Q1" occurs 3 times between timestamps 101 and 700, which is within a 600-second window. Since *threshold* = 3, "Q1" satisfies the required frequency.
Hence, the answer is ["Q1"].

Question - 17
Recent Error Tracker

Given arrays of *timestamps* and *errorCodes* representing logs sorted in non-decreasing order of timestamps, find all error codes that occurred at least *k* times within the last *t* seconds.

The function *getErrorCodes* will take four inputs:
int k: the minimum number of occurrences required
int t: the size of the time window in seconds
int timestamps[n]: array of timestamps
string errorCodes[n]: array of error codes at corresponding timestamps

The function should return a list of strings denoting all error codes that occurred at least *k* times within the last *t* seconds (sorted order). If there is no such error code, return an empty array.

Example
k = 2
t = 10
timestamps = [100, 101, 102, 105, 110]
errorCodes = ["E1", "E2", "E1", "E1", "E2"]

The table below shows the time window during which the error code appeared at least *k* times, along with its corresponding frequency count:

Error Code Table

Time Window	Error Code	Frequency Count
[101, 110]	E1	2
[101, 110]	E2	2

The last *t* seconds form the time window [101, 110].
The error codes "E1" and "E2" each occur 2 times in the time window [101, 110]. Since *k* = 2, both codes meet the frequency requirement.
Hence, the answer is ["E1", "E2"].

Constraints
- $1 \le n, k \le 2 * 10^5$
- 1 ≤ length of *errorCodes[i]* ≤ 10
- $1 \le timestamps[i] \le 10^9$
- $1 \le t \le timestamps[n-1]$
- It is guaranteed that *errorCodes[i]* is an alphanumeric string.
- It is guaranteed that the array *timestamps* is sorted in non-decreasing order.

▼ Input Format for Custom Testing

The first line contains an integer *k*,.
The second line contains an integer *t*.
The third line contains an integer *n*, the size of the array *timestamps*.
The next *n* lines each contain an integer *timestamps[i]*.
The next line contains an integer *n*, the size of the array *errorCodes*.
The next *n* lines each contain a string *errorCodes[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
3       →     k = 3
5       →     t = 5
7       →     timestamps[] size n = 7
1       →     timestamps = [1, 2, 4, 5, 6, 7, 10]
2
4
5
6
7
10
7       →     errorCodes[] size n = 7
E1      →     errorCodes = ["E1", "E2", "E1", "E1", "E2", "E2", "E2"]
E2
E1
E1
E2
E2
E2
```

Sample Output 0

```
E2
```

Explanation

The table below shows the time window during which the error code appeared at least *k* times, along with its corresponding frequency count:

Error Code Table

Time Window	Error Code	Frequency Count
[6, 10]	E1	0
[6, 10]	E2	3

The last *t* seconds form the time window [6, 10].
The error code "E2" occurs 3 times. Since *k* = 3, the code meets the frequency requirement.
Hence, the answer is ["E2"].

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
2       →     k = 2
4       →     t = 4
5       →     timestamps[] size n = 5
1       →     timestamps = [1, 2, 4, 5, 6]
2
4
5
6
5       →     errorCodes[] size n = 5
E1      →     errorCodes = ["E1", "E2", "E1", "E1", "E2"]
E2
E1
E1
E2
```

Sample Output 1

```
E1
```

Explanation

The table below shows the time window during which the error code appeared at least *k* times, along with its corresponding frequency count:

Error Code Table

Time Window	Error Code	Frequency Count
[3, 6]	E1	2
[3, 6]	E2	1

The last *t* seconds form the time window [3, 6].
The error code "E1" occurs 2 times in the time window [3, 6].
Hence, the answer is ["E1"].

Question - 18
Load Spike Detector

Find all the indices *i* (0-based) of the timestamps such that the server load of the *i*th timestamp is greater than twice the average server load.

The function *getHighLoadTimestamps* will take one input:
int load[n]: server loads at timestamps

The function should return an array of integers denoting the indices (0-based) of the timestamps, in increasing order, where *load[i]* > 2 × average server load. If none, return an empty array.

Example
n = 3
load = [1, 2, 9]

The average load is (1 + 2 + 9) / 3 = 4. Only the value at timestamp 2 (0-based indexing) is greater than 2 × 4 = 8.
Hence, the answer is [2].

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le load[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer *n*, the size of the array *load*.
Each of the next *n* lines contains an integer *load[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
5     →  load size[] n = 5
1     →  load = [1, 1, 7, 7, 1]
1
7
7
1
```

Sample Output

```
2
```

```
3
```

Explanation

The average load is (1 + 1 + 7 + 7 + 1) / 5 = 3.4. The values at timestamps 2 and 3 (0-based indexing) are greater than 2 × 3.4 = 6.8.

Hence, the answer is [2, 3].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
4     →  load size[] n = 4
3     →  load = [3, 3, 3, 3]
3
3
3
```

Sample Output

```

```

Explanation

The average load is (3 + 3 + 3 + 3) / 4 = 3. The values at all the timestamps are less than 2 × 3 = 6.

Hence, the answer is [].

Question - 19

Find Missing Log Window

You are given a list of unique integers greater than 0 and possibly out of order. Find the first missing sequence of consecutive positive numbers of at least a given length. Return the starting number of this missing window.

The function *findMissingWindow* will take two inputs:

int timestamp[n]: an array of *n* unique integers starting from 1, representing the array *timestamps*.

int windowSize: the minimum length of the missing consecutive window to find.

The function should return an integer denoting the smallest integer of the first missing contiguous window of length at least *windowSize*.

Example

n = 4

timestamp = [4, 2, 1, 7]

windowSize = 2

The missing timestamps are [3, 5, 6, 8, …]. The first missing contiguous window of length at least *windowSize* is [5, 6].

Hence, the answer is 5.

Constraints

- $1 \le n, windowSize \le 2 * 10^5$
- $1 \le timestamp[i] \le 10^9$
- It is guaranteed that the array *timestamp* contains unique integers.

▼ Input Format For Custom Testing

The first line contains an integer *n*, the size of the array *timestamp*.

Each of the next *n* lines contains an integer *timestamp[i]*.

The next line contains an integer *windowSize*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
3      →   timestamp[] n = 3
2      →   timestamp = [2, 1, 3]
1
3
1      →   windowSize = 1
```

Sample Output

```
4
```

Explanation

The missing timestamps are [4, 5, 6, ...], and the first missing contiguous window of length at least windowSize is [4].

Hence, the answer is 4.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
2      →   timestamp[] n = 2
3      →   timestamp = [3, 6]
6
2      →   windowSize = 2
```

Sample Output

```
1
```

Explanation

The missing timestamps are [1, 2, 4, 5, 7, 8, ...], the first missing contiguous window of length at least *windowSize* is [1, 2].

Hence, the answer is 1.

Question - 20
Process Execution Timeline

Given a list of processes with their start and end times (inclusive), find the total amount of time during which at least one process was running.

The function *getExecutionTime* will take two inputs:
  int start[n]: the start times of the processes
  int end[n]: the end times of the processes

The function should return an integer denoting the total time during which at least one process was running.

Example

n = 3

start = [1, 2, 8]

end = [5, 6, 10]

The table below shows the merged intervals during which at least one process is running.

Execution Timeline Table

Original Intervals	Merged Interval	Total Time

[1, 5], [2, 6]	[1, 6]	6
[8, 10]	[8, 10]	3

- The intervals [1, 5] and [2, 6] overlap, so they are merged into [1, 6].
- The interval [8, 10] does not overlap with any of the previous ones, so it forms a separate block.

Total time = (6 – 1 + 1) + (10 – 8 + 1) = 6 + 3 = 9

Hence, the answer is 9.

Constraints

- $1 \le n \le 2 * 10^5$
- $0 \le start[i] \le end[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *start*.

The next *n* lines each contain an integer *start[i]*.

The next line contains an integer *n*, the size of the array *end*.

The next *n* lines each contain an integer *end[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
2       →     start[] size n = 2
1       →     start[] = [1, 5]
5
2       →     end[] size n = 2
3       →     end[] = [3, 10]
10
```

Sample Output 0

```
9
```

Explanation

The table below shows the merged intervals during which at least one process is running.

Execution Timeline Table

Original Intervals	Merged Interval	Total Time
[1, 3]	[1, 3]	3
[5, 10]	[5, 10]	6

Total time = (3 – 1 + 1) + (10 – 5 + 1) = 3 + 6 = 9

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
3       →     start[] size n = 3
5       →     start[] = [5, 6, 12]
6
```

```
12
3       →     n = 3
8       →     end[] = [8, 9, 13]
9
13
```

Sample Output 1

```
7
```

Explanation

Execution Timeline Table

Original Intervals	Merged Interval	Total Time
[5, 8], [6, 9]	[5, 9]	5
[12, 13]	[12, 13]	2

- The intervals [5, 8] and [6, 9] overlap and are merged into [5, 9].
- The interval [12, 13] does not overlap with any of the previous ones, so it forms a separate block.

Total time = (9 – 5 + 1) + (13 – 12 + 1) = 5 + 2 = 7

Question - 21
Concurrent Event Counter

Given a list of events with their start and end times (inclusive), find the highest number of events that overlap at any moment.

The function *getMaxOverlap* will take two inputs:
  int start[n]: the start times of the events.
  int end[n]: the end times of the events.

The function should return an integer denoting the maximum number of overlapping events at any time.

Example
n = 3
start = [1, 2, 4]
end = [5, 6, 7]

The number of overlapping events over the time period from 1 to 7.

Event Overlap Table

Time	Number of Overlaps	Overlapping Intervals
1	1	[1, 5]
2	2	[1, 5], [2, 6]
3	2	[1, 5], [2, 6]
4	3	[1, 5], [2, 6], [4, 7]
5	3	[1, 5], [2, 6], [4, 7]

6	2	[2, 6], [4, 7]
7	1	[4, 7]

The maximum number of overlapping events is 3.
Hence, the answer is 3.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- 0 ≤ *start[i]* ≤ *end[i]* ≤ 10^9

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *start*.
The next *n* lines each contain an integer *start[i]*.
The next line contains an integer *n*, the size of the array *end*.
The next *n* lines each contain an integer *end[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
2       →     start[] size n = 2
1       →     start[] = [1, 2]
2
2       →     end[] size n = 2
3       →     end[] = [3, 4]
4
```

Sample Output 0

```
2
```

Explanation

Event Overlap Table

Time	Number of Overlaps	Overlapping Intervals
1	1	[1, 3]
2	2	[1, 3], [2, 4]
3	2	[1, 3], [2, 4]
4	1	[2, 4]

The maximum number of overlapping events is 2.

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
3       →     start[] size n = 3
5       →     start[] = [5, 6, 7]
6
7
3       →     end[] size n = 3
8       →     end[] = [8, 9, 10]
```

```
9
10
```

Sample Output 1

```
3
```

Explanation

Event Overlap Table

Time	Number of Overlaps	Overlapping Intervals
5	1	[5, 8]
6	2	[5, 8], [6, 9]
7	3	[5, 8], [6, 9], [7, 10]
8	3	[5, 8], [6, 9], [7, 10]
9	2	[6, 9], [7, 10]
10	1	[7, 10]

The maximum number of overlapping events is 3.

Question - 22
Code Bracket Validator

You are given an array of strings, where each string contains only the characters '(', ')', '{', '}', '[', and ']'. For each string, determine if the string is valid.

A string is valid if:

- Every opening bracket has a matching closing bracket of the same type.
- Brackets close in the correct order.

The function *isValidBrackets* will take one input:
string queries[q]: the string queries

The function should return an array of strings, where each element is "YES" if the corresponding input string is valid, or "NO" otherwise.

Example
q = 4
queries = ["[]{{}}", "([]{}", "{[()]}", "]({})["]

Analysis of each bracket sequence is as follows:

1. "[]{{}}" → All brackets are correctly matched and nested.
2. "**(**[]{}" → Missing closing bracket for '('.
3. "{[()]}" → Brackets are properly nested and matched.
4. "**]**({})[" → Starts with closing bracket ']' without a match.

Hence, the answer is ["YES", "NO", "YES", "NO"].

Constraints

- $1 \le q \le 2 * 10^5$

- 1 ≤ length of *queries[i]* ≤ 2 * 10^5
- It is guaranteed that the sum of the lengths of strings over all the queries does not exceed 2 * 10^5.

▾ Input Format for Custom Testing

The first line contains an integer *q*, the size of the array *queries*.

Each of the next *i* lines (where 0 ≤ *i* < *q*) contains a string, *queries[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN           Function
-----           --------
3          →    queries[] size q = 3
[()]{}     →    queries = ["[()]{}", "{[}]()", "(){}"]
{[}]()
(){}
```

Sample Output 0

```
YES
NO
YES
```

Explanation

Analysis of each bracket sequence is as follows:

1. "[()]{}" → Brackets are correctly nested and all pairs match.
2. "{[}]()" → Incorrect closing: '}' closes before '[' is matched.
3. "(){}" → Brackets are properly matched and in correct order.

▾ Sample Case 1

Sample Input 1

```
STDIN           Function
-----           --------
4          →    queries[] size q = 4
{([])      →    queries = ["{([])", "({[}])", "[[]{()}]", "(({})[)]"]
({[}])
[[]{()}]
(({})[)]
```

Sample Output 1

```
NO
NO
YES
NO
```

Explanation

Analysis of each bracket sequence is as follows:

1. "**{**([])" → Missing closing bracket for '{'.
2. "({[}])" → Incorrect closing: '}' closes before '[' is matched.
3. "[[]{()}]" → All brackets are properly matched and nested.
4. "(({})[)]" → Incorrect closing: ')' closes before '[' is matched.

Question - 23

Log File Analysis

Given timestamps of login attempts on a web server, two login attempts at indices *i* and *j* (where *i*, *j* are 0-based and *i* < *j*) are considered suspicious if they occur within the specified window — that is, if the condition |*timestamps[i] - timestamps[j]*| ≤ *timeWindow* holds.

Find the number of such suspicious login-attempt pairs in the array.

The function *countSuspiciousPairs* will take two inputs:
  int timestamps[n]: the login attempt times in seconds
  int timeWindow: the maximum allowed time difference (in seconds) between two login attempts to consider them suspicious

The function should return a long integer denoting the number of suspicious pairs.

Example
n = 4
timestamps = [52, 35, 43, 62]
timeWindow = 10

The suspicious pairs of indices

*i*th index	*j*th index	|*timestamps[i] - timestamps[j]*|
0	2	|52 - 43| = 9 ≤ 10
0	3	|52 - 62| = 10 ≤ 10
1	2	|35 - 43| = 8 ≤ 10

Hence, the answer is 3.

Constraints
- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ *timestamps[i]*, *timeWindow* ≤ 10^9

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *timestamps*.
Each of the next *n* lines (where 0 ≤ *i* < *n*) contains an integer, *timestamps[i]*.
The last line contains an integer, *timeWindow*.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
3       →      timestamps[] size n = 3
4       →      timestamps = [4, 1, 6]
1
6
3       →      timeWindow = 3
```

Sample Output 0

```
2
```

Explanation

The suspicious pairs of indices

i^{th} index	j^{th} index	*|timestamps[i] - timestamps[j]|*
0	1	|4 - 1| = 3 ≤ 3
0	2	|4 - 6| = 2 ≤ 3

▾ Sample Case 1

Sample Input 1

```
STDIN           Function
-----           --------
4        →      timestamps[] size n = 4
15       →      timestamps = [15, 30, 45, 20]
30
45
20
25       →      timeWindow = 25
```

Sample Output 1

```
5
```

Explanation

The suspicious pairs of indices

i^{th} index	j^{th} index	*|timestamps[i] - timestamps[j]|*
0	1	|15 - 30| = 15 ≤ 25
0	3	|15 - 20| = 5 ≤ 25
1	2	|30 - 45| = 15 ≤ 25
1	3	|30 - 20| = 10 ≤ 25
2	3	|45 - 20| = 25 ≤ 25

Question - 24
Frequent Request Detection

Implement a request deduplication system for an API that may receive duplicate requests due to network retries. Two requests are considered duplicates if their content is identical and they occur within a given time window.

The function *countFrequentRequestIDs* will take three inputs:
string requestIDs[n]: IDs of the requests
int timestamps[n]: the time when the requests occurred (in seconds)
int timeWindow: the window length (in seconds)

The function should return an integer denoting the number of unique request IDs that repeat within any time window of length *timeWindow*.

Example
n = 4
requestIDs = ["authreq001", "authreq002", "authreq001", "authreq002"]
timestamps = [10, 25, 20, 15]

timeWindow = 10

Summary of each unique request ID based on the time window condition:

- "authreq001" appears at 10 and 20, 20 - 10 = 10 ≤ 10 (*timeWindow*), which satisfies the condition
- "authreq002" appears at 15 and 25, 25 - 15 = 10 ≤ 10 (*timeWindow*), which satisfies the condition

Hence, the answer is 2.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ length of *requestIDs[i]* ≤ 2 * 10^5 and the sum of their lengths over all *i* does not exceed 2 * 10^5.
- 1 ≤ *timestamps[i]*, *timeWindow* ≤ 10^9
- It is guaranteed that *requestIDs[i]* consists only of characters in the ranges ascii['a' - 'z', '0' - '9'].

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *requestIDs*.
Each of the next *n* lines (where 0 ≤ *i* < *n*) contains a string, *requestIDs[i]*.
The next line contains an integer *n*, the size of the array *timestamps*.
Each of the next *n* lines (where 0 ≤ *i* < *n*) contains an integer, *timestamps[i]*.
The last line contains an integer, *timeWindow*.

▼ Sample Case 0

Sample Input 0

```
STDIN                Function
-----                --------
3           →        requestIDs[] size n = 3
req1        →        requestIDs = ["req1", "req2", "req1"]
req2
req1
3           →        timestamps[] size n = 3
45          →        timestamps = [45, 35, 40]
35
40
5           →        timeWindow = 5
```

Sample Output 0

```
1
```

Explanation

- "req1" appears at 45 and 40, 45 - 40 = 5 ≤ 5 (*timeWindow*), which satisfies the condition
- "req2" appears only once, so it does not satisfy the condition

▼ Sample Case 1

Sample Input 1

```
STDIN                Function
-----                --------
5           →        requestIDs[] size n = 5
user123     →        requestIDs = ["user123", "user456", "user123", "user789", "user456"]
user456
user123
user789
user456
5           →        timestamps[] size n = 5
10          →        timestamps = [10, 38, 23, 30, 25]
38
23
30
25
15          →        timeWindow = 15
```

Sample Output 1

```
2
```

Explanation

- "user123" appears at 10 and 23 → 23 - 10 = 13 ≤ 15 (*timeWindow*), which satisfies the condition
- "user456" appears at 38 and 25 → 38 - 25 = 13 ≤ 15 (*timeWindow*), which satisfies the condition
- "user789" appears only once, so it does not satisfy the condition

Question - 25
Memory Allocation Tracker

Given arrays *start* and *end* representing the inclusive time intervals of memory allocations, find the earliest time when the maximum number of memory allocations are active simultaneously.

The function *getMaxConcurrentAllocation* will take two inputs:
int start[n]: starting times of allocations
int end[n]: ending times of allocations

The function should return an integer denoting the earliest time when the maximum number of allocations are active simultaneously.

Example
start = [1, 2, 4]
end = [5, 6, 7]

The number of memory allocations at each time is from 1 to 7. The underlined element is the maximum number of overlapping allocations with the earliest time.

Memory Allocation Table

Time	Current number of allocations
1	1
2	2
3	2
<u>4</u>	<u>3</u>
5	3
6	2
7	1

The maximum number of overlapping allocations is 3, which first occurs at time 4, so the answer is 4.

Constraints

- $1 \le n \le 2 * 10^5$
- $0 \le start[i] \le end[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *start*.

The next *n* lines each contain an integer *start[i]*.

The next line contains an integer *n*, the size of the array *end*.

The next *n* lines each contain an integer *end[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
2       →     start[] size n = 2
1       →     start = [1, 2]
2
2       →     end[] size n = 2
3       →     end = [3, 4]
4
```

Sample Output 0

```
2
```

Explanation

Memory Allocation Table

Time	Current number of allocations
1	1
2	2
3	2
4	1

The maximum number of overlapping allocations is 2, which first occurs at time 2.

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
3       →     start[] size n = 3
5       →     start = [5, 6, 7]
6
7
3       →     end[] size n = 3
8       →     end = [8, 9, 10]
9
10
```

Sample Output 1

```
7
```

Explanation

Memory Allocation Table

Time	Current number of allocations
5	1
6	2
7	3

8	3
9	2
10	1

The maximum number of overlapping allocations is 3, which first occurs at time 7.

Question - 26
Maximum of Subarray Minimums

Implement a function that, for an array of integers *arr* and a subarray of size *k*, determines the maximum among the minimums of all contiguous subarrays of size *k*.

The function *maximumSubarrayMinimum* takes the following input:
int arr[n]: an array of integers
int k: the subarray length

The function should return an integer, the largest value amongst all minimums of subarrays of size *k.*

Note: A subarray is an array obtained by deleting several (possibly zero) elements from the beginning and end of the original array.

Example
n = 5
arr = [1, 2, 3, 4, 5]
k = 2

For subarray size *k* = 2, the subarrays are [1, 2], [2, 3], [3, 4], and [4, 5], and their minima are [1, 2, 3, 4]. The final answer is 4, the maximum of these.

Constraints
- $1 \le n \le 10^6$
- $1 \le arr[i] \le 10^9$
- $1 \le k \le n$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *arr*.
Each line *i* of the next *n* lines contains an integer, *arr[i]*.
The last line contains an integer, *k*.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
5      →  arr[] size n = 5
1      →  arr = [1, 2, 3, 1, 2]
2
3
1
2
1      →  k = 1
```

Sample Output

```
3
```

Explanation

Each element of *arr* = [1, 2, 3, 1, 2] is an array of size *k* = 1 and its minimum. The maximum of these minima is 3.

▼ Sample Case 1

Sample Input

```
STDIN     Function
-----     -----
3      →  arr[] size n = 3
1      →  arr = [1, 1, 1]
1
1
2      →  k = 2
```

Sample Output

```
1
```

Explanation
The two subarrays of size *k* = 2 are [1, 1], [1, 1]. Their minima are [1, 1], and the maximum of these is 1.

Question - 27
API Rate-Limiting

You are building a system that distributes API requests across *n* servers. Each server has a certain capacity, given by *capacity[i]*, which is the number of requests it can handle in 1 second. After handling requests, the server's capacity becomes half of what it was before (rounded down).

Implement a function that returns the minimum time required to handle all incoming requests given by *requests*.

The function *calculateSchedulingTime* takes the following inputs:
int capacity[n]: each element denotes the capacity of the i^{th} server
long requests: the number of incoming API requests

The function should return an integer, the minimum time required to process all incoming API requests.

Example
n = 5
capacity = [3, 1, 7, 2, 4]
requests = 15

The optimal solution is:

1. First, the node with capacity = 7 schedules 7 requests in one second. Now, *capacity* = [3, 1, 3, 2, 4] because 7 was reduced to floor(7/2). There are 15 - 7 = 8 remaining requests.
2. Second, the node with capacity = 4 is used. After that, *capacity* = [3, 1, 3, 2, 2]. Remaining requests = 8 - 4 = 4.
3. Third, a node with capacity = 3 is used. Now, *capacity* = [1, 1, 3, 2, 2]. Remaining requests = 4 - 3 = 1.
4. Finally, a node with a capacity of 1 is used to schedule the final request.

Each step requires one second of processing time, so the answer is 4.

Constraints

- $1 \le n \le 10^5$
- $1 \le capacity[i] \le 10^6$
- $1 \le requests \le 10^{12}$
- It is guaranteed that the requests can be scheduled using the given system.

▼ Input Format For Custom Testing

The first line contains an integer *n*, the size of *capacity[]*.

Each of the next i^{th} lines (where $0 \le i < n$) contains an integer, *capacity[i]*, the capacity of the i^{th} server.

The next line contains an integer, *requests*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
5      →   capacity[] size n = 5
2      →   capacity = [2, 1, 5, 3, 1]
1
5
3
1
17     →   requests = 17
```

Sample Output

```
9
```

Explanation

The optimal solution is :

1. After choosing the servers with *capacity* = 2, 5, and 3, a total of 10 requests are scheduled (requiring 3 seconds, one for each server). Now, *capacity* = [1, 1, 2, 1, 1] and there are 17 - 10 = 7 remaining requests.
2. Then choose the server with *capacity* = 2 to schedule 2 requests (requiring 1 second). After that, *capacity* = [1, 1, 1, 1, 1], and there are 7 - 2 = 5 requests remaining.
3. All 5 servers are used to schedule one request each (requiring 5 seconds).

The total time required is (3+1+5) = 9 seconds.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
4      →   capacity[] size n = 4
3      →   capacity = [3, 1, 4, 2]
1
4
2
3      →   requests = 3
```

Sample Output

```
1
```

Explanation

Choose the server with *capacity* = 3. All 3 requests are scheduled, and the total time required is 1 second.

Question - 28
Trend Analytics Monitoring

You are building a system that analyzes performance metrics over time, where each trend is given by *trends[i]*. One feature involves identifying bitonic trends, i.e., sequences that first rise (non-decreasing) and then fall (non-increasing).

Implement a function to find the length of the longest subarray that is bitonic in nature.

The function *longestBitonicTrend* takes the following input:

int trends[n]: an array of integers

The function should return an integer, the length of the longest bitonic subarray.

Example
trends = [10, 8, 9, 15, 12, 6, 7]

[8, 9, 15, 12, 6] is the longest bitonic subarray. The non-decreasing subarray is [8, 9, 15], and the non-increasing portion is [15, 12, 6].

Hence, the answer is 5.

Constraints

- 1 ≤ *n* ≤ 10^5
- 1 ≤ *trends[i]* ≤ 10^9

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of elements in *trends*.
Each line *i* of the *n* subsequent lines (where 0 ≤ *i* < *n*) contains an integer, *trends[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
6        →    trends[] size n = 6
5        →    trends = [5, 1, 2, 1, 4, 5]
1
2
1
4
5
```

Sample Output

```
3
```

Explanation
[1, 2, 1] is one of the longest bitonic subarrays. [1, 4, 5] is also bitonic and has the same length. It is non-decreasing through [1, 4, 5], and the non-increasing portion is [5].

Hence, the answer is 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
5        →    trends[] size n = 5
9        →    trends = [9, 7, 6, 2, 1]
7
6
2
1
```

Sample Output

```
5
```

Explanation
The longest bitonic subarray is [9, 7, 6, 2, 1]. The non-decreasing subarray is [9], and the non-increasing portion is the entire array.

Hence, the answer is 5.

Question - 29

Bullish Trend Detection

You are building a backend service for a stock analytics platform. You have an array named *stockPrices* where *stockPrices[i]* denotes the price of the stock in the i^{th} month. A bullish trend of size *k* is an interval of *k* months such that the stock prices increase strictly throughout those months.

Implement a function that calculates the number of bullish trends of size *k*.

The function *calculateBullishTrends* takes the following inputs:

int stockPrices[n]: the stock prices for *n* months

int k: the analysis parameter

The function should return an integer, the number of bullish trends of size *k*.

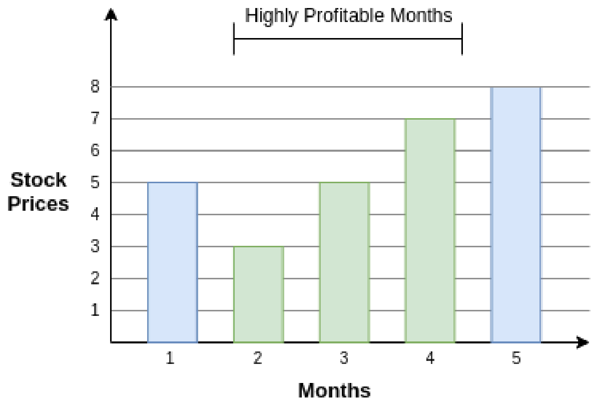
Note: If the interval length is 1, each subarray of length 1 is highly profitable.

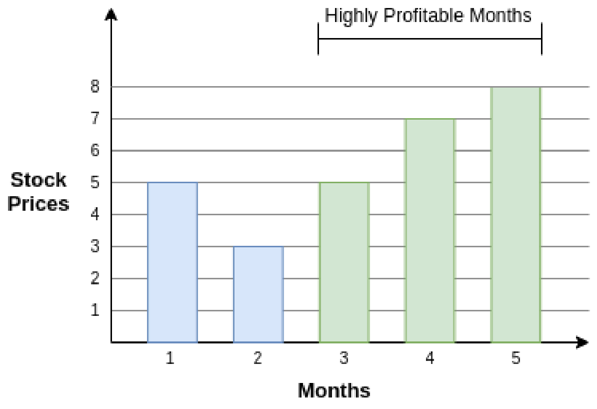
Example

stockPrices = [5, 3, 5, 7, 8]

k = 3

These are the intervals of *k* months in which the stock prices are strictly increasing:





Hence, the answer is 2.

Constraints

- $1 \le k \le n \le 2 * 10^5$
- $1 \le stockPrices[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of elements in the array *stockPrices*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer *stockPrices[i]*.

The last line contains an integer *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
6       →    stockPrices[] size, n = 6
1       →    stockPrices = [1, 2, 3, 3, 4, 5]
2
3
```

```
3
4
5
3          →    k = 3
```

Sample Output

```
2
```

Explanation

- Months 1 to 3: [1, 2, 3]
- Months 4 to 6: [3, 4, 5]

Hence, the answer is 2.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4          →    stockPrices[] size, n = 4
1          →    stockPrices = [1, 2, 3, 4]
2
3
4
4          →    k = 4
```

Sample Output

```
1
```

Explanation
Since *k* is equal to *n*, we can only make one group consisting of all the months. Since this is strictly increasing, the answer is 1.

Question - 30
Cloud Function Optimization

You are running a list of cloud functions where each function has a cost given by the array *functionCosts*, and you have a starting budget *k*. You must go through the list from start to end.

You earn 1 point for each function you run (if you can afford it). You are allowed to skip exactly one function without spending any budget or earning a point.

Implement a function to find the maximum number of functions you can run.

The function *maximumPoints* takes the following inputs:
  int k: the initial budget
  int functionCosts[n]: the costs of each function

The function should return an integer, the maximum number of points that can be earned, given the budget constraint and the ability to skip at most one function.

Example
k = 14
n = 5
functionCosts = [2, 4, 1, 8, 6]

Some possible ways of spending the budget :

- Running 5 functions without skipping any: cost = 2 + 4 + 1 + 8 + 6 = 21 > *k*.
- Running 5 functions while skipping the 4th function: cost = 2 + 4 + 1 + 6 = 13 ≤ *k*, points = 4, as functions 1, 2, 3, and 5 were run.

It can be proven that you cannot collect more than 4 points. Hence, the answer is 4.

Constraints

- $1 \le k \le 10^9$
- $1 \le n \le 10^5$
- $1 \le functionCosts[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer *k*.
The second line contains an integer *n*, the size of the array *functionCosts*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *functionCosts[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
10    →  k = 10
5     →  n = 5
5     →  functionCosts = [5, 2, 3, 1, 4]
2
3
1
4
```

Sample Output

```
4
```

Explanation

Some possible ways of spending the budget :

- Running 5 functions without skipping any, cost = 5 + 2 + 3 + 1 + 4 = 15 > *k*.
- Running 5 functions and skipping the 4th function, cost = 5 + 2 + 3 + 4 = 14 > *k*.
- Running 5 functions and skipping the 1st function, cost = 2 + 3 + 1 + 4 = 10 ≤ *k*, points = 4, as functions 2, 3, 4, and 5 were run.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
15    →  k = 15
6     →  n = 6
3     →  functionCosts = [3, 2, 6, 4, 6, 1]
2
6
4
6
1
```

Sample Output

```
4
```

Explanation

Some possible ways of spending the budget :

- Running 6 functions without skipping any, cost = 3 + 2 + 6 + 4 + 6 + 1 = 22 > *k*.
- Running 5 functions and skipping the 3rd function, cost = 3 + 2 + 4 + 6 = 15 ≤ *k*, points = 4, as functions 1, 2, 4, and 5 were run.

Question - 31

Minimum Memory Compressions

You are building a memory management system. Each element in an array represents a memory block of size *memoryBlocks[i]*.

Periodically, you run a compression operation that clears out a set of memory blocks:

- In one operation, you can select any subset of memory blocks whose sum of $2^{memoryBlocks[i]}$ values is equal to 2^p for some non-negative integer *p*.
- Once removed, those blocks are no longer considered in the array.

Implement a function that determines the minimum number of compression operations required to free all memory blocks.

The function *minimumCompressions* takes the following input:

int memoryBlocks[n]: an array representing the sizes of the memory blocks.

The function should return an integer, the minimum number of compression operations required.

Example

n = 5

memoryBlocks = [1, 1, 3, 2, 3]

The array can be emptied in 2 operations :

1. Remove *memoryBlocks[0], memoryBlocks[1]* and *memoryBlocks[3]* as their sum ($2^1 + 2^1 + 2^2 = 8$) equals 2^3. The array becomes [3, 3].
2. Remove *memoryBlocks[0]* and *memoryBlocks[1]* as their sum ($2^3 + 2^3 = 16$) equals 2^4. The array becomes empty.

It can be shown that the minimum number of operations cannot be less than 2.

Constraints

- $1 \le n \le 10^5$
- $0 \le memoryBlocks[i] \le 10^6$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *memoryBlocks*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains integer *memoryBlocks[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     memoryBlocks[] size n = 4
1        →     memoryBlocks = [1, 1, 3, 5]
1
3
5
```

Sample Output

```
3
```

Explanation

The array can be emptied in 3 operations :

1. Remove *memoryBlocks[1]* and *memoryBlocks[2]* as their sum ($2^1 + 2^1 = 4$) equals 2^2. The array becomes [3, 5].
2. Remove *memoryBlocks[1]*. The array becomes [5].
3. Remove *memoryBlocks[1]*. The array becomes empty.

It can be shown that the minimum number of operations cannot be less than 3.

Hence, the answer is 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     memoryBlocks[] size n = 4
0        →     memoryBlocks = [0, 2, 0, 1]
2
0
1
```

Sample Output

```
1
```

Explanation
The array can be emptied in 1 operation :

1. Remove *memoryBlocks[0], memoryBlocks[1], memoryBlocks[2], memoryBlocks[3]* as their sum ($2^0 + 2^2 + 2^0 + 2^1 = 8$) equals 2^3. The array becomes empty.

It can be shown that the minimum number of operations cannot be less than 1.
Hence, the answer is 1.

Question - 32
Repository Issue Resolution

A developer is assigned to work on *n* repositories, each with a set of open issues that must be resolved so the repository is fully maintained.

You are given:

- *resolved[i]:* the number of issues already resolved in the i^{th} repository
- *required[i]:* the number of issues that must be resolved for the i^{th} repository to be marked as maintained
- *k:* the number of additional issues the developer can fix across all repositories

Implement a function that distributes the remaining *k* issue resolutions to maximize the number of repositories that can be marked as maintained.

The function *maxMaintainedRepos* takes the following inputs:
  int resolved[n]: issues already resolved for each repository
  int required[n]: required issue resolutions for each repository
  int k: number of additional issues the developer can fix

The function should return an integer representing the maximum number of repositories that can be marked as maintained after optimally distributing the remaining fixes.

Example
n = 2
resolved = [2, 4]
required = [4, 5]
k = 1

It is optimal to fix the second repository (index = 1), since the number of additional issues the developer can fix is less than what is required for the first (index = 0). Only the issues of the second repo can only resolved with the given *k*.

Hence, the answer is 1.

Constraints

- $1 \le n \le 10^5$
- $0 \le resolved[i], required[i], k \le 10^9$

▼ **Input Format For Custom Testing**

The first line contains an integer, *n,* the number of elements in *resolved*.

Each of the next *n* lines contains one integer, *resolved[i]*.

The next line contains the integer, *n,* the number of elements in *required*.

Each of the next *n* lines contains one integer, *required[i].*

The next line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        →    resolved[] size n = 3
24       →    resolved[] = [24, 27, 0]
27
0
3        →    required[] size n = 3
51       →    required[] = [51, 52, 100]
52
100
100      →    k = 100
```

Sample Output

```
2
```

Explanation

Here, *resolved* = [24, 27, 0] and *required* = [51, 52, 100], so needed resolutions are [27, 25, 100].

The optimal distribution of resolutions is 27 + 25 = 52 questions among the first two repositories. It would take all *k* = 100 additional resolutions to maintain the third repo.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        →    resolved[] size n = 3
14       →    resolved[] = [14, 35, 23]
35
23
3        →    required[] size n = 3
47       →    required[] = [47, 83, 59]
83
59
200      →    k = 200
```

Sample Output

```
3
```

Explanation

Here *resolved* = [14, 35, 23] and *required =* [47, 83, 59], so needed resolutions are [33, 48, 36], a total of 117.

Since *k* = 200 resolutions are possible, all repositories can be maintained.

Question - 33
Project Sync

You are managing *link_nodes* distributed code repositories, numbered from 1 to *link_nodes*. Some repositories are already in sync via direct version-control links. A repository is considered synced with another if they are linked either directly or indirectly.

You can perform the reassignment of existing links any number of times (possibly zero):

- Remove one existing link and use it to connect any other pair of repositories.

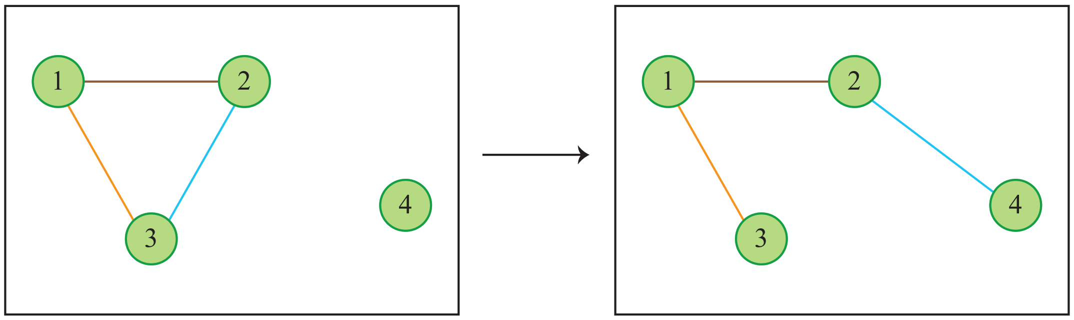
Implement a function that determines the minimum number of link reassignments needed to ensure all repositories are part of a single synced system. If it is not possible to sync all repositories using the available links, return -1.

The function *minReassignments* takes the following inputs:

int link_nodes: the number of repositories
int link_from[link_edges]: the starting points of the current direct sync links
int link_to[link_edges]: the ending points of the current direct sync links

The function should return an integer denoting the minimum number of reassignments required to connect all repositories into one synced system, or -1 if it is not possible.

Example:
link_nodes = 4
link_edges = 3
link_from = [1, 1, 3]
link_to = [2, 3, 2]



There are 4 repositories (*link_nodes* = 4) with 3 direct sync links. The initial links are:

- Repository 1 is synced with repositories 2 and 3
- Repository 2 is synced with repositories 1 and 3
- Repository 3 is synced with repositories 1 and 2
- Repository 4 is not connected to any other repository.

To fully sync all repositories, one existing link (say between repositories 2 and 3) can be reassigned to connect repository 2 with repository 4. All repositories are now part of a single synced system. Thus, the minimum number of reassignments required is 1.

Constraints

- $2 \le link\_nodes \le 2 * 10^5$
- $1 \le link\_edges \le 3 * 10^5$
- $1 \le link\_from[i] \le link\_nodes$
- $1 \le link\_to[i] \le link\_nodes$
- For each *i*, *link_from[i]* is connected to *link_to[i],* and the link is bidirectional.
- There are no self-loops or multiple links between any two repositories.

▼ Input Format For Custom Testing

The first line contains two space-separated integers, *link_nodes* and *link_edges*.
Each line *i* of the *link_edges* subsequent lines (where $0 \le i < link\_edges$) contains 2 space-separated integers, *link_from[i]* and *link_to[i]*.

▼ Sample Case 0

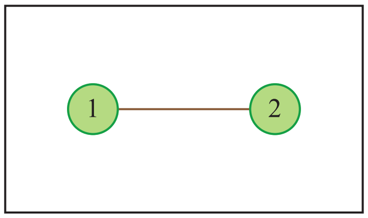
Sample Input For Custom Testing

```
STDIN          Function
-----          --------
2 1      →     link_nodes = 2, link_edges = 1
1 2      →     link_from[] = [1], link_to[] = [2]
```

Sample Output

```
0
```

Explanation



All the repositories are synced. Hence, the answer is 0.

▼ Sample Case 1

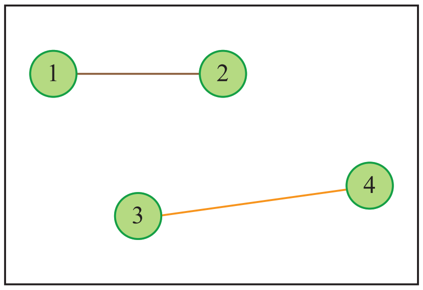
Sample Input For Custom Testing

```
STDIN         Function
-----         --------
4 2     →     link_nodes = 4, link_edges = 2
1 2     →     link_from[] = [1, 2], link_to[] = [3 , 4]
3 4
```

Sample Output

```
-1
```

Explanation



In this case, it is not possible to sync all the repositories. Hence, the answer is -1.

Question - 34
Optimal Assignment

You are given a collection of items and their associated costs, where each item can have multiple cost options. There are *numItems* total items, labeled from 0 to *numItems* - 1.

You have two integer arrays that define the available cost options, *itemId* and *cost*, both of length *n*, where:

- *itemId[i]* is the label of the item for the i^{th} cost entry
- *cost[i]* is the cost associated with that item

Implement a function that returns the minimum total cost to select exactly one cost entry for each item. If any item does not have a cost entry, return -1.

The function *minTotalCost* takes the following inputs:

int numItems: the total number of items labeled from 0 to *numItems* - 1
int itemId[n]: an array of item labels for each cost entry
int cost[n]: an array containing the corresponding costs for each entry

The function should return a long integer representing the minimum total cost to select all the items, or -1 if it is impossible to select every item.

Example

numItems = 3

n = 4

itemId = [2, 0, 1, 2]

cost = [8, 7, 6, 9]

An optimal way of selecting exactly one cost entry for each item is :

1. For item 0, there is only one available cost, which is 7.
2. For item 1, there is also only one available cost, which is 6.
3. For item 2, it is better to choose the first cost entry (8) instead of the fourth (9).

Therefore, the minimum total cost is 7 + 6 + 8 = 21. Hence, the answer is 21.

Constraints

- $1 \le$ *numItems, n* $\le 5 * 10^5$
- $0 \le$ *itemId[i]* < *numItems*
- $1 \le$ *cost[i]* $\le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *numItems*.

The next line contains an integer, *n*, which denotes the number of elements in *itemId[]*.

Each line *i* of the *n* subsequent lines (where 0 ≤ *i* < *n*) contains an integer, *itemId[i]*.

The next line contains an integer, *n*, which denotes the number of elements in *cost[]*.

Each line *i* of the *n* subsequent lines (where 0 ≤ *i* < *n*) contains an integer, *cost[i]*.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
2     →   numItems = 2
5     →   itemId[] size n = 5
0     →   itemId = [0, 1, 0, 1, 1]
1
0
1
1
5     →   cost[] size n = 5
4     →   cost = [4, 74, 47, 744, 7]
74
47
744
7
```

Sample Output

```
11
```

Explanation

An optimal way of selecting exactly one cost entry for each item is :

1. For item 0, it is better to choose the 1st cost entry (4) instead of the 3rd (47).
2. For item 1, it is better to choose the 5th cost entry (7) instead of the 2nd (74) and 4th (744).

The minimum total cost is 4 + 7 = 11.

▼ Sample Case 1

Sample Input

```
STDIN     Function
-----     -----
2      →  numItems = 2
2      →  itemId[] size n = 2
1      →  itemId = [1, 1]
1
2      →  cost[] size n = 2
4      →  cost = [4, 7]
7
```

Sample Output

```
-1
```

Explanation

There are no cost entries for item 0.

Hence, the answer is -1.

Question - 35
Subarray Partition

Implement a function that determines the maximum number of non-overlapping, contiguous subarrays in an array *arr* of length *n*, that satisfy the following constraints:

- Each subarray must contain at least *minLen* elements.
- The sum of elements in each subarray must be greater than or equal to *threshold*.

The function *numSubarrays* takes the following inputs:

int arr[n]: the array to partition
int minLen: the minimum number of elements required in each valid subarray
long threshold: the minimum sum required for a subarray to be considered valid

The function should return an integer denoting the maximum number of valid subarrays that can be formed without any overlap and satisfying the given constraints.

Example

n = 6
arr = [5, 7, 9, 12, 10, 13]
minLen = 2
threshold = 15

An optimal way of splitting the array into valid subarrays is (using 0-based indexing) :

- The first valid subarray is from index 1 to 3 ([7, 9, 12]), with a sum of 28, which satisfies the condition.
- The second valid subarray is from index 4 to 5 ([10, 13]), with a sum of 23, which also satisfies the condition.

Hence, the answer is 2.

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$
- $1 \le minLen \le n$
- $1 \le threshold \le 10^{14}$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the size of *arr*.
Each of the next *i* lines contains an integer, *arr[i]*.

The next line contains an integer, *minLen*.

The next line contains an integer, *threshold*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
7      →   n = 7
3      →   arr = [3, 3, 2, 5, 3, 2, 4]
3
2
5
3
2
4
2      →   minLen = 2
7      →   threshold = 7
```

Sample Output

```
2
```

Explanation

An optimal way of splitting the array into valid subarrays is (using 0-based indexing) :

- The first valid subarray is from index 2 to 3 ([2, 5]), with a sum of 7, which satisfies the condition.
- The second valid subarray is from index 4 to 6 ([3, 2, 4]), with a sum of 9, which also satisfies the condition.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
5      →   n = 5
1      →   arr = [1, 2, 3, 4, 5]
2
3
4
5
3      →   minLen = 3
20     →   threshold = 20
```

Sample Output

```
0
```

Explanation

No valid subarrays can be formed because the threshold value is greater than the sum of all elements in the array.

Question - 36
Balance Audit

Implement a function that determines the members in a list of *n* transactions, given by the array *transactions*, have the most negative balance. Each transaction consists of: [*sender, receiver, amount*]. The *sender* loses that amount, and the *receiver* gains it. The *amount* is always positive in each transaction.

The function *getMinBalanceMembers* takes the following input:

string transactions[n][3]: a 2D array representing the transactions

The function should return the following:

- Return a list of all keys associated with the most negative net value, sorted alphabetically.
- If no key has a negative net value, then return ["Nobody has a negative balance"].

Example

n = 6

transactions = [["Alex", "Blake", "2"], ["Blake", "Alex", "2"], ["Casey", "Alex", "5"], ["Blake", "Casey", "7"], ["Alex", "Blake", "4"], ["Alex", "Casey", "4"]]

There are 6 transactions, as shown in the table below:

sender	*receiver*	*amount*
Alex	Blake	2
Blake	Alex	2
Casey	Alex	5
Blake	Casey	7
Alex	Blake	4
Alex	Casey	4

- Alex receives (2+5) and sends (2+4+4), so the net balance is 7 - 10 = -3.
- Blake receives (2+4) and sends (2+7), so the net balance is 6 - 9 = -3.
- Casey receives (7+4) and sends 5, so the net balance is 11 - 5 = 6.

Hence, the answer (sorted in alphabetical order) is ["Alex", "Blake"] since both of them have the most negative balance of -3 among all members.

Constraints

- $1 \le n \le 2 * 10^5$
- *transactions[i][2]* represents an integer between 1 and 1000 (inclusive) as a string.
- 1 ≤ |*transactions[i][0]*|, |*transactions[i][1]*| ≤ 20, where |*s*| denotes the length of string *s*
- The first character of *transactions[i][0]* and *transactions[i][1]* is a capital English letter.
- Every character of *transactions[i][0]* and *transactions[i][1]* except the first one is a lowercase English letter.
- *transactions[i][0]* ≠ *transactions[i][1]*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of rows in *transactions*.

The second line contains an integer, 3, the number of elements in *transactions[i]*.

Each line *i* of the *n* subsequent lines (where 0 ≤ *i* < *n*) contains *3* space-separated strings: *transactions[i][0], transactions[i][1],* and *transactions[i][2]: sender, receiver*, and *amount*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN                 Function
-----                 --------
5                →    transactions[] size n = 5
3                →    transactions[i] size = 3 (always)
Alex Blake 5     →    transactions = [['Alex', 'Blake', '5'], ['Blake', 'Alex', '3'], ['Casey', 'Alex', '7'],
['Casey', 'Alex', '4'], ['Casey', 'Alex', '2']]
Blake Alex 3
Casey Alex 7
Casey Alex 4
Casey Alex 2
```

Sample Output

```
Casey
```

Explanation

There are 5 transactions, as shown in the table below:

sender	*receiver*	*amount*
Alex	Blake	5
Blake	Alex	3
Casey	Alex	7
Casey	Alex	4
Casey	Alex	2

- Alex receives (3+7+4+2) and sends 5, so the net balance is 16 - 5 = 11.
- Blake receives 5 and sends 3, so the net balance is 5 - 3 = 2.
- Casey receives 0 and sends (7+4+2), so the net balance is 0 - 13 = -13.

Hence, the answer is ["Casey"] since only Casey has the most negative balance of -13 among all members.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN               Function
-----               --------
5              →    transactions[] size n = 5
3              →    transactions[i] size = 3 (always)
Blake Alex 7   →    transactions[] = [['Blake' 'Alex' '7], ['Blake' 'Alex' '3], ['Alex' 'Blake' '4], ['Blake'
'Alex' '1'], ['Alex', 'Blake', '7']
Blake Alex 3
Alex Blake 4
Blake Alex 1
Alex Blake 7
```

Sample Output

```
Nobody has a negative balance
```

Explanation

There are 5 transactions, as shown in the table below:

sender	*receiver*	*amount*
Blake	Alex	7
Blake	Alex	3
Alex	Blake	4
Blake	Alex	1
Alex	Blake	7

- Alex receives (7+3+1) and sends (4+7), so the net balance is 11 - 11 = 0.
- Blake receives (4+7) and sends (7+3+1), so the net balance is 11 - 11 = 0.

Hence, the answer is ["Nobody has a negative balance"] since nobody has a net negative balance.

Question - 37
Interaction Pair Retrieval

In a logging system, an API tracks every microservice interaction with the server. An array named *services* of length *n* exists, where *services[i]* represents the ID of the microservice interacting with the server at the i^{th} moment.

To analyze interaction patterns, the system computes a cross-interaction log:

- The system generates all possible ordered service pairs (*services[i], services[j]*) for all $0 \le i, j < n$.
- These service interaction pairs are then sorted lexicographically.

Implement a function that returns the k^{th} (1-based) interaction pair.

The function *getKthInteraction* takes the following inputs:
int services[n]: the list of service IDs
long k: the 1-based index of the interaction to retrieve

The function should return a list of two integers representing the k^{th} lexicographically smallest interaction pair.

Note: An element *(a, b)* is lexicographically smaller than element *(x, y)* if it satisfies:

- $a < x$, or
- $a = x$ and $b < y$.

Example
$n = 3$
services = [3, 1, 2]
$k = 7$

All possible interactions = [(3,3), (3,1), (3,2), (1,3), (1,1), (1,2), (2,3), (2,1), (2,2)].
Lexicographically sorted order = [(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)].

The 7th interaction pair is [3, 1]. Hence, the function should return [3, 1].

Constraints

- $1 \le n \le 10^5$
- $1 \le services[i] \le 10^9$
- $1 \le k \le n^2$

▼ Input Format For Custom Testing

The first line contains an integer, *n*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *services[i]*.
The last line contains an integer, *k.*

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
2        →    services[] size n = 2
4        →    services = [4, 1]
1
3        →    k = 3
```

Sample Output

```
4
1
```

Explanation

All possible interactions = [(4, 4), (4, 1), (1, 4), (1, 1)]

Lexicographically sorted order = [(1, 1), (1, 4), (4, 1), (4, 4)].

The 3rd interaction pair is (4, 1). Hence, the function should return [4, 1].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        →    services[] size n = 3
2        →    services = [2, 2, 1]
2
1
5        →    k = 5
```

Sample Output

```
2
1
```

Explanation

All possible interactions = [(2, 2), (2, 2), (2, 1), (2, 2), (2, 2), (2, 1), (1, 2), (1, 2), (1, 1)]

Lexicographically sorted order = [(1, 1), (1, 2), (1, 2), (2, 1), (2, 1), (2, 2), (2, 2), (2, 2), (2, 2)].

The 5th interaction pair is (2, 1). Hence, the function should return [2, 1].

Question - 38

Equivalence Checker

You are given two string arrays, *sourceNames* and *targetNames*, each containing *n* filenames. Your task is to determine whether each pair of corresponding filenames can be made identical through at most one bulk character replacement operation.

A bulk character replacement operation allows you to:

- Choose either the source filename or the target filename from a pair
- Select any single character in the chosen filename
- Replace all occurrences of that character with a different character

Implement a function that processes these filename pairs and returns the results.

The function *checkFilenameEquivalence* takes two parameters:

string sourceNames[n]: an array of source filenames

string targetNames[n]: an array of target filenames

The function should return an array of strings where each element corresponds to the result for a filename pair. Return "YES" if the pair can be made identical using the allowed operation, otherwise return "NO".

Example

n = 3

sourceNames = ["aaa", "abbc", "zyz"]

targetNames = ["bbb", "cccc", "zyx"]

Compare the filenames at each index:

- *i* = 0: *sourceNames[0]* = "aaa", *targetNames[0]* = "bbb"
  We can change all the occurrences of 'a' to 'b' in the first string in this case. Therefore, the answer is "YES".
- *i* = 1: *sourceNames[1]* = "abbc", *targetNames[1]* = "cccc"
  There is no way in which we can make these strings identical. Therefore, the answer is "NO".
- *i* = 2: *sourceNames[2]* = "zyz", *targetNames[2]* = "zyx"
  We can replace the only occurrence of 'x' in the second string with 'z' to make them identical. Therefore, the answer is "YES".

Hence, the answer corresponding to each pair of strings is ["YES", "NO", "YES"].

Constraints

- $1 \le n \le 10^3$
- $1 \le |sourceNames|, |targetNames| \le 10^3$, where $|s|$ denotes the length of string *s*
- All the strings consist of lowercase English letters only.

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of elements in *sourceNames*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a string *sourceNames[i]*.
The next line contains an integer *n*, the number of elements in *targetNames*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a string *targetNames[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
2        →     sourceNames[] size n = 2
ac       →     sourceNames = ["ac", "pqqrr"]
pqqrr
2        →     targetNames[] size n = 2
ca       →     targetNames = ["ca", "prrrr"]
prrrr
```

Sample Output

```
NO
YES
```

Explanation

- *i* = 0: *sourceNames[0]* = "ac", *targetNames[0]* = "ca"
  There is no way in which we can make these strings identical. Therefore, the answer is "NO".
- *i* = 1: *sourceNames[1]* = "pqqrr", *targetNames[1]* = "prrrr"
  We can replace all the occurrences of 'q' in the first string with 'r' to make them identical. Therefore, the answer is "YES".

Hence, the answer corresponding to each pair of strings is ["NO", "YES"].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
2        →     sourceNames[] size n = 2
a        →     sourceNames = ["a", "aaabbb"]
aaabbb
2        →     targetNames[] size n = 2
b        →     targetNames = ["b", "abbbbb"]
abbbbb
```

Sample Output

```
YES
```

```
NO
```

Explanation

- *i* = 0: *sourceNames[0]* = "a", *targetNames[0]* = "b"
  We can change the only occurrence of 'a' to 'b' in the first string to make them identical. Therefore, the answer is "YES".
- *i* = 1: *sourceNames[1]* = "aaabbb", *targetNames[1]* = "abbbbb"
  There is no way in which we can make these strings identical. Therefore, the answer is "NO".

Hence, the answer corresponding to each pair of strings is ["YES", "NO"].

Question - 39
Maximum Ones

Implement a function that finds the maximum possible number of '1's in the string *s* after performing at most *k* operations.

In one operation:

- Choose an index *i* (where 0 ≤ *i* < *length of s* - 1) and update the character at position *i* as follows: *s[i] = max(s[i], s[i + 1])*.

The function *getMaximumOnes* will take two inputs:
string s: the string
int k: the maximum number of operations allowed

The function should return an integer representing the maximum possible number of '1's in the string *s.*

Example
s = "10110"
k = 1

An optimal sequence of at most k operations

Operation Number	*s* before	Choosen Index (0-based)	*s* after
1	1<u>0</u>110	Choose the index *i* = 1	1<u>1</u>110

Hence, the maximum number of ones is 4.

Constraints

- 1 ≤ length of *s* ≤ $2 * 10^5$
- 0 ≤ *k* ≤ length of *s*
- It is guaranteed that string *s* contains only '0's and '1's.

▼ Input Format for Custom Testing

The first line contains the string *s*.
The next line contains an integer *k*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
00011     →   s = "00011"
2         →   k = 2
```

Sample Output 0

```
4
```

Explanation

An optimal sequence of at most k operations

Operation Number	*s* before	Choosen Index (0-based)	*s* after
1	00<u>0</u>11	Choose the index *i* = 2	00<u>1</u>11
2	0<u>0</u>111	Choose the index *i* = 1	0<u>1</u>111

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
000000000 →    s = "000000000"
3         →    k = 3
```

Sample Output 1

```
0
```

Explanation

The string will remain unchanged no matter where the operation is performed.

Question - 40
Array Split

Given an array *arr* of size *n*, split it into two non-empty subarrays at any index *i* (where 1 ≤ *i* < *n*). Let the two subarrays be *arr[0...i-1]* and *arr[i...n-1]*.

Implement a function that determines the maximum possible sum of the number of distinct elements in each subarray.

The function *getMaxSum* will take one input:

int arr[n]: an array of integers

The function should return an integer denoting the maximum possible sum of the counts of distinct integers in subarrays formed after splitting.

Example

n = 5

arr = [3, 1, 4, 1, 5]

Distinct count sums for all valid split positions

Split Index *i*	Left Subarray *arr[0..i−1]*	Right Subarray *arr[i..n−1]*	Distinct elements (Left + Right)	Sum
1	[3]	[1, 4, 1, 5]	1 + 3	4
2	[3, 1]	[4, 1, 5]	2 + 3	5
3	[3, 1, 4]	[1, 5]	3 + 2	5
4	[3, 1, 4, 1]	[5]	3 + 1	4

Hence, the maximum possible sum of the count of distinct integers obtained by splitting the array is 5.

Constraints

- $2 \le n \le 2 * 10^5$
- $1 \le arr[i] \le n$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN                    Function
-----                    --------
5            →           arr[] size n = 5
1            →           arr = [1, 3, 1, 2, 1]
3
1
2
1
```

Sample Output 0

```
4
```

Explanation

Distinct count sums for all valid split positions

Split Index *i*	Left Subarray *arr[0..i−1]*	Right Subarray *arr[i..n−1]*	Distinct elements (Left + Right)	Sum
1	[1]	[3, 1, 2, 1]	1 + 3	4
2	[1, 3]	[1, 2, 1]	2 + 2	4
3	[1, 3, 1]	[2, 1]	2 + 2	4
4	[1, 3, 1, 2]	[1]	3 + 1	4

▼ Sample Case 1

Sample Input 1

```
STDIN                    Function
-----                    --------
8            →           arr[] size n = 8
7            →           arr = [7, 6, 5, 6, 8, 6, 6, 1]
6
5
6
8
6
6
1
```

Sample Output 1

```
6
```

Explanation

Distinct count sums for all valid split positions

Split Index *i*	Left Subarray *arr[0..i−1]*	Right Subarray *arr[i..n−1]*	Distinct elements (Left + Right)	Sum
1	[7]	[6, 5, 6, 8, 6, 6, 1]	1 + 4	5
2	[7, 6]	[5, 6, 8, 6, 6, 1]	2 + 4	6
3	[7, 6, 5]	[6, 8, 6, 6, 1]	3 + 3	6
4	[7, 6, 5, 6]	[8, 6, 6, 1]	3 + 3	6
5	[7, 6, 5, 6, 8]	[6, 6, 1]	4 + 2	6
6	[7, 6, 5, 6, 8, 6]	[6, 1]	4 + 2	6
7	[7, 6, 5, 6, 8, 6, 6]	[1]	4 + 1	5

Question - 41
Closest API Latency Spikes

You are building a monitoring tool to track API latency behavior. Given an array *latencies* of distinct integers where each element represents the response time (in milliseconds) of an API at a particular time, the gap between any pair of elements *(i<j)* is given by |*latencies[i] - latencies[j]*|.

Implement a function to identify the pairs of latency values that are closest to each other.

The function *closestLatencySpikes* will take the following input*:*
int latencies[n]: An array of distinct integers representing API latencies.

The function should return a 2-dimenional array, where each element is an array of size 2 containing the elements *(latencies[i], latencies[j])* such that *latencies[i] < latencies[j]* and the absolute difference between *latencies[i]* and *latencies[j]* is minimal, with all pairs sorted in ascending order.

Example
latencies = [6, 2, 4, 10]

The minimal absolute difference is *2,* and the pairs with that difference are (2,4*)* and (4,6). Within the pairs, the elements are ordered, and then the pairs themselves are in ascending order.

Thus, the function should return [[2, 4], [4, 6]]

Constraints
- $2 \le n \le 10^5$
- $-2 \times 10^6 \le$ *latencies[i]* $\le 2 * 10^6$
- The *latencies* array contains no duplicate elements.

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *latencies*.
Each of the next *n* lines contains an integer, *latencies[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
```

```
4       →   latencies[] size n = 4
4       →   latencies = [4, 2, 1, 3]
2
1
3
```

Sample Output 0

```
1 2
2 3
3 4
```

Explanation

The minimum absolute difference between any two elements in the array is *1*, and there are three pairs with this difference.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
-----       --------
4       →   latencies[] size n = 4
4       →   latencies = [4, -2, -1, 3]
-2
-1
3
```

Sample Output 1

```
-2 -1
3 4
```

Explanation

The minimum absolute difference between any two elements in the array is *1*, and there are two pairs with this difference.

Question - 42
Sort Bug Frequencies

You are helping quality assurance engineers process bug reports from automated testing logs. Each log has an integer bug code. A test session may contain duplicate bug codes if the same issue was triggered multiple times.

To prioritize debugging:

- Less frequent bugs are more important as they may represent edge cases.
- If two bugs occur the same number of times, the lower-numbered bugs are prioritized.

Implement a function to sort the errors in order of decreasing importance.

The function *sortBugReportFrequencies* will take the following input:
int bugs[]: An integer array with each element denoting a bug code of an occurring bug.

The function should return an array of integers sorted in order of decreasing importance.

Example
bugs = [8, 4, 6, 5, 4, 8].

Item Code	Frequency
8	2
4	2

6	1
5	1

Bugs with frequency 1 will come before the bugs with frequency 2.

- (6, 5) comes before (8, 4, 4, 8), which results in *bugs* = [6, 5, 8, 4, 4, 8].

In the case of the same frequency, ties are broken by bug codes themselves.

- 5 comes before 6, and 4 comes before 8, which results in *bugs* = [5, 6, 4, 4, 8, 8].

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le bugs[i] \le 10^6$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of bugs*[]*.
The next *n* lines each contain an integer, *bugs[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
5         →    bugs[] size n = 5
3         →    bugs  = [3, 1, 2, 2, 4]
1
2
2
4
```

Sample Output 0

```
1
3
4
2
2
```

Explanation

- frequency of 1: [1], [3], [4]
- frequency of 2: [2, 2]

Sort the array by frequency and then by bug codes in ascending order.

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-----     --------
10    →   bugs[] size n = 10
8     →   bugs = [8, 5, 5, 5, 5, 1, 1, 1, 4, 4]
5
5
5
5
1
1
1
4
4
```

Sample Output 1

```
8
4
4
1
1
1
5
5
5
5
```

Explanation

- frequency 1: [8]
- frequency 2: [4, 4]
- frequency 3: [1, 1, 1]
- frequency 4: [5, 5, 5, 5]

Question - 43
Git Commits

You are working with a Git-based version control system. Each commit in a repository has a unique integer ID representing its timestamp, and the list of commit IDs is given in ascending order. The commit history is complete and consistent.

You are auditing this history and want to test how many consecutive commits can be temporarily hidden (e.g., by a UI filter) such that the remaining visible commits still allow someone to deduce which commits were hidden uniquely. You are allowed to hide a contiguous block of consecutive commits.

Implement a function to maximize the number of commits hidden while ensuring the surrounding commit history clearly defines the hidden range.

The function *getMaxConsecutiveHidden* takes the following input:
 int commits[]: unique integers in ascending order

The function should return an integer, which maximizes the number of commits hidden.

Example
commits = [1, 3, 4, 5, 6, 9]

After hiding the maximum number of consecutive commits, the array becomes [1, 3, _, _, 6, 9]. It is clear that the missing commits are 4 and 5.
Hence, the optimal answer is 2.

Constraints

- $1 \leq n \leq 2 * 10^5$
- $1 \leq commits[i] \leq 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in commits.
Each of the next *n* lines contains an integer commits*[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
5        →    commits[] size n = 5
1        →    commits = [1, 2, 3, 4, 5]
2
3
```

```
4
5
```

Sample Output

```
4
```

Explanation
The first 4 commits can be hidden, and the array becomes [_, _, _, _, 5]. The missing commits can be deduced to be 1, 2, 3, and 4.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
4        →    commits[] size n = 4
3        →    commits = [3, 5, 7, 9]
5
7
9
```

Sample Output

```
0
```

Explanation
If any element is hidden, the array cannot be uniquely reconstructed.

Question - 44
Minimum Operations to Palindrome

A palindromic string reads the same forward and backward. For example, "0", "111", and "010" are palindromes, but "001" and "11101" are not.

You are optimizing a logging system that stores logs as binary strings. However, due to a sorting bug, some logs are jumbled. For performance analytics, your system requires these binary logs to be palindromic for easier indexing and retrieval.

You can perform the following operation any number of times:

- swap any two different bits in the log string

Implement a function to calculate the minimum number of operations required to convert the binary string to a palindrome.

The function *getMinOperations* takes the following input:
string logData: A binary string consisting of characters '0' and '1'.

The function should return an integer, which is the minimum number of operations required to turn it into a palindrome. If it is impossible, return -1.

Example
logData = "0100101"

The following optimal operations are applied sequentially:

- Swap characters at 0-based indices (3, 4), that is 0100101→ 0101001
- Swap characters at 0-based indices (0, 1), that is 0101001→ 1001001

A minimum of 2 operations is required to make *logData* a palindrome.

Constraints

- 1 ≤ length of *logData* ≤ $2 * 10^5$
- *logData[i]* is an element of {'0', '1'}

▼ Input Format For Custom Testing

The only line contains the string *logData*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
101000   →     logData = "101000"
```

Sample Output

```
1
```

Explanation
To form a palindrome from the given string, swap the characters at indices (2, 5), transforming it from '101000' to '100001'.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
1110     →     logData = "1110"
```

Sample Output

```
-1
```

Explanation
There is no way to make the string a palindrome.

Question - 45
Minimizing Task Resource Consumption

In a task scheduling system, you are given an array of *n* integers *resourceConsumption,* where each integer represents the resource consumption value of a task.

The objective is to perform the following operation until no tasks are left. In each iteration:

- Select the task with the lowest resource consumption from the *resourceConsumption*.
- Remove the task along with its adjacent tasks in the array.

Implement a function that computes the total resource consumption of all the selected tasks.

The function *selectLeastResourceTasks* takes the following input:
  int resourceConsumption[n]: each integer represents the resource consumption of a task

The function should return an integer representing the total resource consumption of all the selected tasks.

Note: If there is more than 1 task with the same lowest resource consumption, then select the task with the smallest index.

Example
n = 4
resourceConsumption = [4, 3, 2, 1]

Simulating the defined operation:

- Select the task with the lowest resource consumption (1) and discard it along with its adjacent task (2). So, *resourceConsumption* = [4, 3].
- Select the next lowest (3) and discard the adjacent task (4). So, *resourceConsumption* = [].

The total resource consumption of the selected tasks is 1 + 3 = 4.

Hence, the answer is 4.

Constraints

- 3 ≤ *n* ≤ 2000
- 1 ≤ *resourceConsumption[i]* ≤ 10^5

▼ Input Format For Custom Testing

The first line contains a single integer, *n*, the number of resources.
The next *n* lines contain an integer resourceConsumption[i].

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----      ---------
7    →     n = 7
6    →     resourceConsumption[] = [6, 4, 9, 10, 34, 56, 54]
4
9
10
34
56
54
```

Sample Output

```
68
```

Explanation
Simulating the defined operation:

- First, select the task with the least resource consumption: 4. Its adjacent tasks, 6 and 9, are discarded along with it. The updated array *resourceConsumption* = [10, 34, 56, 54].
- Next, the new minimum is 10, so it and its adjacent task 34 are removed. Now *resourceConsumption* = [56, 54].
- Finally, the remaining minimum is 54, and its adjacent 56 are discarded.

The resource consumption of the selected tasks is: 4 + 10 + 54 = 68.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN        Function
-----        ---------
8      →     n = 8
132    →     resourceConsumption[] = [132, 45, 65, 765, 345, 243, 75, 67]
45
65
765
345
243
75
67
```

Sample Output

```
1120
```

Explanation
Simulating the defined operation:
From:

- [132, 45, 65, 765, 345, 243, 75, 67], select 45, remove [132, 45, 65]
- [765, 345, 243, 75, 67], select 67, remove [75, 67]
- [765, 345, 243], select 243, remove [345, 243]
- [765] select 765, remove [765]

45 + 67 + 243 + 765 = 1120

Question - 46
Android In-App Upgrades

In an Android app, there are *n* in-app upgrades listed. The cost of the *i*th upgrade is given by *upgradeCosts[i],* and it enhances the app's functionality by a factor of 2^i. You are given a limited amount of currency represented by an integer, *budget*.

Implement a function that selects a combination of upgrades such that the total enhancement is maximized without exceeding the available *budget*.

The function *optimizeInAppUpgrades* takes the following inputs:
int upgradeCosts[n]: the costs of each in-app upgrade
int budget: the in-app currency available to spend

The function should return the maximum enhancement that can be obtained, modulo (10^9+7)

Example
n = 5
upgradeCosts = [10, 20, 14, 40, 50]
budget = 70.

Some valid upgrade combinations are:

- Select upgrades 0, 1, and 2 → total cost = 10 + 20 + 14 = 44 → enhancement = $2^0 + 2^1 + 2^2$ = 1 + 2 + 4 = 7
- Select upgrades 0 and 4 → cost = 10 + 50 = 60 → enhancement = $2^0 + 2^4$ = 1 + 16 = 17
- Select upgrades 1 and 4 → cost = 20 + 50 = 70 → enhancement = $2^1 + 2^4$ = 2 + 16 = 18
- Select upgrades 2 and 4 → cost = 14 + 50 = 64 → enhancement = $2^2 + 2^4$ = 4 + 16 = 20 ← (maximum)

Out of all possible combinations, the maximum enhancement that can be obtained is 20, and 20 modulo (10^9+7) = 20.

Hence, the answer is 20.

Constraints
- $1 \le n \le 10^5$
- $1 \le upgradeCosts[i] \le 10^5$
- $0 \le budget \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *upgradeCosts*.
Each of the next *n* lines contains an integer describing *upgradeCosts[i]*.
The next line contains an integer, *budget*, the amount of money.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
3    →   n =  3
3    →   upgradeCosts = [3, 4, 1]
4
```

```
1
8    →   budget = 8
```

Sample Output

```
7
```

Explanation

All the upgrades can be purchased since the total cost is within the developer's allocated budget.
The total impact from the selected upgrades would be $2^0 + 2^1 + 2^2 = 7$, and 7 modulo $(10^9 + 7) = 7$.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
5    →   n = 5
19   →   upgradeCosts = [19, 78, 27, 18, 20]
78
27
18
20
25   →   budget = 25
```

Sample Output

```
16
```

Explanation
The optimal strategy is to purchase only the final upgrade, yielding an impact of $2^4 = 16$, and 16 modulo $(10^9 + 7) = 16$.

Question - 47
Throttle Window Optimization

In a web infrastructure, a system logs *n* API requests per second based on their response times. These logs are stored in an array *responseTimes*, where each entry denotes the time taken by a request.

To mitigate overload, a throttling mechanism is applied as follows:

- Select the request with the minimum response time, for priority analysis.
- Due to batching behavior in the system, the selected request and its adjacent requests are removed from the logs permanently.

This process is repeated until all requests are removed.

Implement a function to determine the total cumulative response times of all the selected requests.

The function *totalResponseTime* takes the following input:
int responseTimes[n]: the response times of each request logged by the server

The function should return the sum of the response times of all requests selected throughout the throttling process.

Note: If multiple requests have the same minimum response time, select the one with the lowest index.

Example
n = 4
responseTimes = [4, 3, 2, 1]

Simulating the throttling mechanism:

- Select the request with the minimum response time (1) and discard it along with its adjacent request with response time (2). So, *responseTimes:* [4, 3]
- Next, choose 3 (minimum in remaining), and discard it along with the adjacent request with response time (4). So, *responseTimes:* []

Thus, the total sum of response times of all the selected requests is 1 + 3 = 4.
Hence, the answer is 4.

Constraints

- $3 \le n \le 2000$
- $1 \le$ *responseTimes[i]* $\le 10^5$

▼ Input Format For Custom Testing

The first line contains a single integer, *n*, representing the number of integers in the array *responseTimes.*
The next *n* lines contain integers, where the *i*th integer denotes the value at position *responseTimes[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----      ---------
7    →     responseTimes[] size is 7
6    →     responseTimes = [6, 4, 9, 10, 34, 56, 54]
4
9
10
34
56
54
```

Sample Output

```
68
```

Explanation

Simulating the throttling mechanism:

- From [6, 4, 9, 10, 34, 56, 54], select the request with the lowest response time, 4. Remove it and its neighbors, [6, 4, 9].
- From [10, 34, 56, 54], the lowest response time is 10. Remove it and its neighbor, [10, 34].
- From [56, 54], select 54. Remove it and its neighbor, [56, 54], and the array is empty.

The sum of selected response times is 4 + 10 + 54 = 68.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
-----     ---------
8    →    responseTimes[] size is 8
132  →    responseTimes = [132, 45, 65, 765, 345, 243, 75, 67]
45
65
765
345
243
75
67
```

Sample Output

```
1120
```

Explanation
Simulating the throttling mechanism:

- From [132, 45, 65, 765, 345, 243, 75, 67], select 45, remove [132, 45, 65].
- From [765, 345, 243, 75, 67], select 67, remove [75, 67]

- From [765, 345, 243], select 243, remove [345, 243]
- From [765], select and remove 765.

The sum of selected response times is: 45 + 67 + 243 + 765 = 1120.

Question - 48
Maximize VM Rental Revenue

In a cloud computing platform, there are *n* types of virtual machines (VMs). Each VM type has a certain number of instances available for rent, represented by the array *vmStock*.

Each time a VM of a certain type is rented, the customer pays a fee equal to the number of instances currently available for that type. There are *m* different customers arriving one by one, and each customer always chooses the most expensive VM available at that moment, i.e., the VM type with the most instances left.

Implement a function to calculate the total revenue the platform can earn after serving all rental requests.

The function *maximizeRentalRevenue* takes the following inputs:
int vmStock[n]: the number of VMs of each type available initially
int m: the number of customers requesting to rent a VM

The function should return a long integer denoting the total revenue that can be earned after serving all customers' requests.

Example
n = 3
vmStock = [1, 2, 4]
m = 4

There are 3 different types of VMs available with 4 customer requests.

Simulation of Purchases

Customer	VM type	Cost	Remaining
Initial state			[1, 2, 4]
1	3	4	[1, 2, 3]
2	3	3	[1, 2, 2]
3	2 (or 3)	2	[1, 1, 2]
4	3	2	[1, 1, 1]

The revenue earned is 4 + 3 + 2 + 2 = 11, so the answer is 11.

Constraints
- $1 \le n \le 10^5$
- $1 \le vmStock[i] \le 10^6$
- $1 \le m \le 10^6$
- The sum of *vmStock[i]* for all $1 \le i \le n$ will always be greater than *m*.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *vmStock*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *vmStock[i]*.

The next line contains an integer, *m,* the number of customers in the queue.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4         →    vmStock[] size n = 4
2         →    vmStock = [2, 1, 1, 3]
1
1
3
4         →    m = 4
```

Sample Output

```
8
```

Explanation

There are 4 different types of VMs available with 4 customer requests.

Simulation of Purchases

Customer	VM type	Cost	Remaining
Initial state			[2, 1, 1, 3]
1	4	3	[2, 1, 1, 2]
2	4	2	[2, 1, 1, 1]
3	1	2	[1, 1, 1, 1]
4	any	1	[1, 1, 1]

The revenue earned is 3 + 2 + 2 + 1 = 8.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
3         →    vmStock[] size n = 3
10        →    vmStock = [10, 10, 11]
10
11
3         →    m = 3
```

Sample Output

```
31
```

Explanation

There are 3 different types of VMs available with 3 customer requests.

Simulation of Purchases

Customer	VM type	Cost	Remaining
Initial state			[10, 10, 11]
1	3	11	[10, 10, 10]

2	3 (any)	10	[10, 10, 9]
3	1 (or 2)	10	[10, 9, 9]

Thus, the total revenue earned is: 11 + 10 + 10 = 31.

Question - 49
Minimize Decompression Steps

In a backend system, a list of *n* identifiers represented by the integer array *ids* of length *n,* is used to represent data in a compressed form. The objective is to ensure that all identifiers are odd numbers for compatibility with other services.

To achieve this, you can perform the following decompression operation an arbitrary number of times (possibly zero):

- Choose any even number *x* from the list and replace all occurrences of *x* with *x* / 2.

Implement a function that determines the minimum number of steps required to make every number in the list odd.

The function *minimizeDecompressionSteps* takes the following input:
int ids[n]: a list of integers representing compressed numeric identifiers

The function should return the minimum number of steps required to make all identifiers odd.

Example
n = 6
ids = [4, 2, 4, 3, 6, 12]

An optimal set of operations

Operation	Select x	Transformed list of IDs
1	4	[**2,** 2, **2,** 3, 6, 12]
2	12	[2, 2, 2, 3, 6, **6**]
3	2	[**1, 1, 1**, 3, 6, 6]
4	6	[1, 1, 1, **3, 3, 3**]

All numbers are now odd. Hence, the answer is 4.

Constraints

- 1 ≤ n ≤ 10^5
- 1 ≤ $ids[i]$ ≤ 10^9

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *ids*.
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer describing *ids[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
```

```
6        →    ids[] size n = 6
2        →    ids = [2, 4, 6, 8, 10, 12]
4
6
8
10
12
```

Sample Output

```
6
```

Explanation

An optimal way of applying the decompression operation:

An optimal set of operations

Operation	Select x	Transformed list of IDs
1	10	[2, 4, 6, 8, **5**, 12]
2	12	[2, 4, 6, 8, 5**, 6**]
3	8	[2, 4, 6, **4**, 5, 6]
4	4	[2, **2**, 6, **2**, 5, 6]
5	2	[**1, 1,** 6, **1,** 5, 6]
6	6	[1, 1, **3**, 1, 5, **3**]

All numbers are now odd. Hence, the answer is 6.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
5        →    ids[] size n = 5
12       →    ids = [12, 4, 24, 6, 5]
4
24
6
5
```

Sample Output

```
5
```

Explanation

An optimal set of operations

Operation	Select x	Transformed list of IDs
1	24	[12, 4, **12**, 6, 5]
2	12	[**6**, 4, 6, **6**, 5]
3	6	[**3**, 4, **3, 3**, 5]
4	4	[**3**, 4, **3, 3**, 5]
5	2	[3, **1**, 3, 3, 5]

All numbers are now odd. Hence, the answer is 5.

Question - 50
Minimize URL Compression

In a URL shortening service, the URL is given as a string *url* consisting of lowercase English letters.

The task is to extract a subsequence of exactly *k* characters from the *url* that minimizes the URL compression, defined as the difference between the length of the subsequence and the number of distinct characters in the subsequence.

Implement a function to compute the minimum possible compression for any subsequence of length *k*.

The function *minimizeURLCompression* takes the following inputs:
string url: a string representing the original URL code
int k: the desired length of the subsequence

The function should return the minimum possible compression in any subsequence of length *k*.

Note: The subsequence of a string is a string obtained by deleting any set of characters from the original string while retaining the order of remaining characters.

Example
url = "ooxoxo"
k = 4

One optimal subsequence of length exactly *k* is "ooxx". The length of the subsequence is 4, and it contains two distinct characters, 'o' and 'x'. The URL compression is 4 - 2 = 2.

Hence, the answer is 2.

Constraints

- $2 \leq k \leq \text{length}(url) \leq 10^5$
- String *url* consists of lowercase English letters only.

▼ Input Format For Custom Testing

The first line contains a string, *url*.
The next line contains an integer *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
aaaabbbb  →  url = "aaaabbbb"
5         →  k = 5
```

Sample Output

```
3
```

Explanation

One optimal subsequence of length exactly *k* is "aabbb". It is 5 characters long and has two distinct characters, 'a' and 'b'. The compression is 5 - 2 = 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
abcdeabcde →  url = "abcdeabcde"
5          →  k = 5
```

Sample Output

```
0
```

Explanation

One optimal subsequence of length exactly *k* is "abcde". Its length is 5 and it has 5 distinct characters, 'a', 'b', 'c', 'd', and 'e'. Its compression is 5 - 5 = 0.

Question - 51
Compression Loss

In a deep learning model, a binary feature map is represented by an array *featureMap* consisting of 0s and 1s.

To speed up inference, the model compresses the feature map by repeatedly applying a reduction operation:

- If the current *featureMap* has *k* (*k > 1*) elements, generate *newFeatureMap[i] = featureMap[i] & featureMap[i+1]* for *0 ≤ i < k - 1*. Replace *featureMap* with *newFeatureMap*. Here, '&' denotes the bitwise AND operation.
- Repeat the process until a single element remains.

Implement a function to calculate the compression loss, defined as the sum of the original array and all intermediate arrays formed during reduction.

The function *computeCompressionLoss* takes the following input:
int featureMap[n]: an array of binary values representing the feature map before compression

The function should return the sum of the original array and all arrays generated during the reduction process.

Example
Consider *n = 4, arr = [0, 0, 1, 1].*

The reduction process proceeds as follows:

- Step 0: *featureMap* = [0, 0, 1, 1] → sum = 2.
- Step 1: Applying the reduction operation: *featureMap* = [0 & 0, 0 & 1, 1 & 1] = [0, 0, 1] → sum = 1.
- Step 2: Applying the reduction operation: *featureMap* = [0 & 0, 0 & 1] = [0, 0] → sum = 0.
- Step 3: Applying the reduction operation: *featureMap* = [0 & 0] = [0] → sum = 0.
- The size of the array becomes 1, so no more operations are performed.

Thus, the total sum of all the arrays generated during the reduction process (2 + 1 + 0 + 0) = 3.
Hence, the answer is 3.

Constraints

- $1 \le n \le 10^6$
- *featureMap[i]* is either 0 or 1, for all *0 ≤ i < n*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *featureMap*.
Each line *i* of the next *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *featureMap[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     featureMap[] size n = 4
1        →     featureMap = [1, 0, 1, 0]
0
1
0
```

Sample Output

```
2
```

Explanation

The reduction process proceeds as follows:

- Step 0: *featureMap* = [1, 0, 1, 0] → sum = 2.
- Step 1: Applying the reduction operation: *featureMap* = [1 & 0, 0 & 1, 1 & 0] = [0, 0, 0] → sum = 0.
- Step 2: Applying the reduction operation: *featureMap* = [0 & 0, 0 & 0] = [0, 0] → sum = 0.
- Step 3: Applying the reduction operation: *featureMap* = [0 & 0] = [0] → sum = 0.
- The size of the array becomes 1, hence no more operations are performed

Thus, the sum of all the arrays generated during the reduction process (2 + 0 + 0 + 0) = 2.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
5        →     featureMap[] size n = 5
1        →     featureMap = [1, 1, 1, 0, 0]
1
1
0
0
```

Sample Output

```
6
```

Explanation

The reduction process proceeds as follows:

- Step 0: *featureMap* = [1, 1, 1, 0, 0] → sum = 3.
- Step 1: Applying the reduction operation: *featureMap* = [1 & 1, 1 & 1, 1 & 0, 0 & 0] = [1, 1, 0, 0] → sum = 2.
- Step 2: Applying the reduction operation: *featureMap* = [1 & 1, 1 & 0, 0 & 0] = [1, 0, 0] → sum = 1.
- Step 3: Applying the reduction operation: *featureMap* = [1 & 0, 0 & 0] = [0, 0] → sum = 0.
- Step 4: Applying the reduction operation: *featureMap* = [0 & 0] = [0] → sum = 0.
- The size of the array becomes 1, hence no more operations are performed

The sum of all the arrays generated during the reduction process (3 + 2 + 1 + 0 + 0) = 6.

Question - 52
Optimal Score Band

In a machine learning optimization system, you are given an array *modelScores* of size *n*, where *modelScores[i]* represents the performance score of the i^{th} model after a training cycle.

Implement a function that determines the size of the largest subsequence of models such that their scores form a continuous value band, i.e., the absolute difference between any two consecutive scores after sorting is at most 1.

The function *findLargestScoreBand* takes the following input:
int modelScores[n]: the performance scores for each model

The function should return an integer, the size of the largest subsequence that satisfies the given requirements.

Note: A subsequence of an array is formed by deleting zero or more elements without changing the order of the remaining elements.

Example
n = 5
modelScores = [8, 5, 4, 8, 4]

Any single-element subsequence forms a value band, so [8], [5], and [4] qualify. Value bands of size 2 are [4, 4], [5, 4] sorted = [4, 5], and [8, 8]. There is one subsequence of size 3: [5, 4, 4] sorted = [4, 4, 5]. The largest subsequence has size 3, so the function should return 3.

Constraints

- $1 \le n \le 10^5$
- $1 \le modelScores[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *modelScores*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *modelScores[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN    Function
-----    --------
4    →    modelScores[] size n = 4
4    →    modelScores = [4, 13, 2, 3]
13
2
3
```

Sample Output 0

```
3
```

Explanation 0
The largest subsequence that forms a value band is [4, 2, 3], sorted = [2, 3, 4].

▼ Sample Case 1

Sample Input 1

```
STDIN    Function
-----    --------
5     →    modelScores[] size n = 5
10    →    modelScores = [10, 12, 1, 10, 4]
12
1
10
4
```

Sample Output 1

```
2
```

Explanation 1
The largest subsequence that forms a value band is [10, 10].

Question - 53
Frequent Transactions

A financial transaction monitoring system uses an array called *transactions* of length *n* to represent the transaction amounts.

Implement a function to reorder the transactions in ascending order based on their frequency, and in case of a tie, by their amounts in ascending order.

The function *sortTransactions* takes the following input:
int transactions[n]: individual transaction amounts

The function should return a sorted list of transaction amounts based on frequency, with ties broken by amount.

Example
n = 6
transactions = [4, 5, 6, 5, 4, 3]

Frequency Map:

- Transactions with values 3 and 6 occur once.
- Transactions with values 4 and 5 occur twice.

Sorting the transactions as per the specified rule:

1. Values with frequency 1 sorted in ascending order: [3, 6].
2. Values with frequency 2 sorted in ascending order: [4, 4, 5, 5].

Hence, the final answer is [3, 6, 4, 4, 5, 5].

Constraints

- $1 \le n \le 2 \times 10^5$
- $1 \le transactions[i] \le 10^6$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of *transactions[]*.
The next *n* lines each contain an integer, *transactions[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function
-----         --------
5        →    transactions[] size n = 5
3        →    transactions = [3, 1, 2, 2, 4]
1
2
2
4
```

Sample Output 0

```
1
3
4
2
2
```

Explanation

- Transaction values with frequency 1 (sorted in ascending order): [1], [3], [4]
- Transaction values with frequency 2 (sorted in ascending order): [2, 2]

Hence, the answer is [1, 3, 4, 2, 2].

▼ **Sample Case 1**

Sample Input 1

```
STDIN     Function
-----     --------
10    →   transactions[] size n = 10
8     →   transactions = [8, 5, 5, 5, 5, 1, 1, 1, 4, 4]
5
5
5
5
1
1
1
4
4
```

Sample Output 1

```
8
4
4
1
1
1
5
5
5
5
```

Explanation

- Transaction values with frequency 1 (sorted in ascending order): [8]
- Transaction values with frequency 2 (sorted in ascending order): [4, 4]
- Transaction values with frequency 3 (sorted in ascending order): [1, 1, 1]
- Transaction values with frequency 4 (sorted in ascending order): [5, 5, 5, 5]

Hence, the answer is [8, 4, 4, 1, 1, 1, 5, 5, 5, 5].

Question - 54
Distributed System Recovery

Your team manages a distributed system of *network_nodes* nodes. A major incident occurs, and the primary server with ID *company* begins recovery operations.

Each node is connected via bidirectional links. Due to the incident, many systems are offline, and your goal is to bring systems back up in a specific order based on the following rules:

1. Start from the main node (*company*); it is already online.
2. Bring systems back up in order of proximity (shortest number of hops from (*company*).
3. If multiple systems are equally distant, recover the one with the lower node number first.
4. Ignore isolated nodes — if a system is unreachable, it cannot be recovered yet.

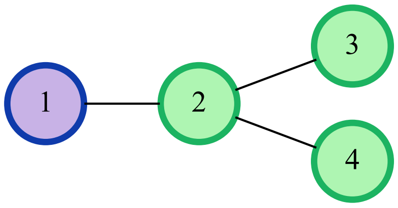
The function *recoverNetwork* takes the following four inputs:
int network_nodes: the number of nodes in the network

int network_from[n]: the first node where there is a bidirectional edge
int network_to[n]: the second node where there is a bidirectional edge
int company: the node from which the recovery starts

Implement a function that returns an array of node numbers in the order they should be recovered, excluding the starting node itself.

Example
network_nodes = 4
network_from = [1, 2, 2]
network_to = [2, 3, 4]
company = 1.

The company is located in node 1, and the edges are between nodes 1 and 2, nodes 2 and 3, and nodes 2 and 4, like so:



In this case, the nodes are visited based on the following logic:

- The closest node, node 2, is visited first. It is 1 unit away.
- The next-closest nodes are visited next. Node 3 and Node 4 are both 2 units distance.
  - In this case, the lower-numbered node is prioritized: visit node 3 first, then node 4.

The answer is [2, 3, 4].

Constraints

- $2 \le network\_nodes \le 10^5$
- $1 \le n \le min((network\_nodes \times (network\_nodes - 1)) / 2, 10^5)$
- $1 \le network\_from[i], network\_to[i], company \le n$
- $network\_from[i] \ne network\_to[i]$

▼ Input Format For Custom Testing

The first line contains two space-separated integers: *network_nodes,* denoting the number of nodes and *n,* the number of bidirectional edges.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains two space-separated integers, *network_from[i]* and *network_to[i].*
The last line contains an integer, *company*.

▼ Sample Case 0

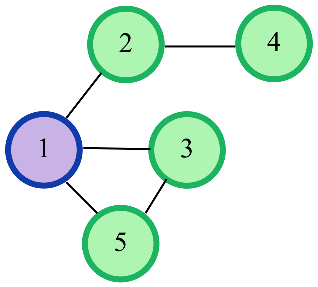
Sample Input For Custom Testing

```
STDIN     Function
-----     --------
5 5   →   network_nodes = 5, n = 5
1 2   →   network_from = [1, 1, 2, 3, 1], network_to = [2, 3, 4, 5, 5]
1 3
2 4
3 5
1 5
1     →   company = 1
```

Sample Output

```
2
3
5
4
```

Explanation



Nodes 2, 3, and 5 are all 1 unit of distance away from the company. These are visited based on priority in ascending order, so [2, 3, 5]. Node 4 is 2 units of distance, so it is visited next.
The final order is [2, 3, 5, 4].

▼ Sample Case 1

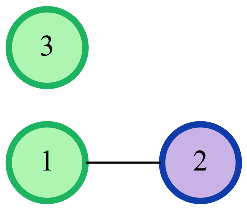
Sample Input For Custom Testing

```
STDIN     Function
-----     --------
3 1   →   network_nodes = 3, n = 1
1 2   →   network_from = [1], network_to = [2]
2     →   company = 2
```

Sample Output

```
1
```

Explanation



Node 1 is 1 unit of distance from the company. Node 3 is not accessible because there are no edges connecting it to the company's node. Therefore, the answer is [1].

Question - 55
SEO Input Processing

You are working on a lightweight search engine indexing module, and one of the early tasks is to filter out non-informative words that appear too frequently, often called stop words. These words are so common across documents that they dilute search relevance.

Implement a function that finds all the words that occur in the string *text* at least *k* times.

The function *findStopWords* will take two inputs:
  string text: a string containing words of letters in the English alphabet separated by space, " "
  int k: the minimum number of occurrences of a word required to be considered as a stop word

The function should return an array of strings, the stop words in order of their first occurrence in the string.

Example
text = "a mouse is smaller than a dog but a dog is stronger"
k = 2

The list of stop words that occur at least *k* = 2 times is ["a", "is", "dog"]. "a" occurs 3 times, "is" and "dog" both occur 2 times. No other word occurs at least 2 times. The answer is in order of the first appearance in *text*.

Constraints

- *text* has at most 50000 characters.
- Every character in *text* is either an English lowercase letter or a space.
- *text* starts and ends with a letter. No two consecutive characters are spaces, i.e., *text* is a valid sentence.
- There will be at least one stop word in the text.
- 1 ≤ *k* ≤ 18

▼ Input Format Format for Custom Testing

The first line contains the string *text*.
The second line contains the integer *k*.

▼ Sample Case 0

Sample Input 0

```
STDIN                                                                  Function
-----                                                                  --------
the brown fox jumps over the brown dog and runs away to a brown house → text
2                                                                    → k
```

Sample Output 0

```
the
brown
```

Explanation

"the" occurs 2 times and "brown" occurs 3 times. These words are returned in the order of their first occurrence in the text.

▼ Sample Case 1

Sample Input 1

```
STDIN                                                                  Function
-----                                                                  --------
foo bar foo baz foo → text
3                                                                    → k
```

Sample Output 1

```
foo
```

Explanation

"foo" occurs 3 times.

Question - 56
Efficient Drone Descent

You are programming a delivery drone to fly through a vertical 4 × 4 grid of city sectors. The drone starts its descent from any sector in the top row and must reach the bottom row. Each grid cell *city[i][j]* represents the power cost to fly through that sector due to factors like wind, obstacles, and elevation. Each time the drone passes through the cell *(i, j)*, its power reduces by *city[i][j]* units.

The drone can descend by any of the following moves:

- Begin at any cell in the first row.
- For each move, go from cell *(i, j)* in the i^{th} row and j^{th} column to any of these cells in the next row: *(i + 1, j - 1), (i + 1, j)*, or *(i + 1, j + 1)*.
- The drone must finish the traversal in the last row.

Implement a function that finds the maximum power level the drone can have after reaching the last row if the drone starts with an initial power of 100 units.

The function *maxPower* will take the following input:

int city[4][4]: a matrix of integers representing the power cost of each sector

The function should return an integer representing the maximum power level left after reaching the last row.

Note: The final power can be negative.

Example

city = [[10, 20, 30, 40],
[60, 50, 20, 80],
[10, 10, 10, 10],
[60, 50, 60, 50]]

Possible paths:

0-based indexing is used.

- (0, 0) - (1, 1) - (2, 2) - (3, 3)
- (0, 1) - (1, 2) - (2, 2) - (3, 1)

For the first path, power left = 100 - 10 - 50 - 10 - 50 = -20

For the second path, power left = 100 - 20 - 20 -10 - 50 = 0

Here, 0 is the maximum power possible at the end of the traversal.

Constraints

- 0 ≤ *city[i][j]* < 100

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of rows in the matrix, *city* (always 4).

The second line contains an integer *m*, the number of columns in the matrix, *city* (always 4).

Each of the next 4 lines contains 4 space-separated integers denoting the power cost for each sector in *city*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4          →   city[][] size n = 4
4          →   city[i] size m = 4
4 6 14 21
17 0 5 5
4 41 22 3
2 51 6 0
```

Sample Output

```
90
```

Explanation

Possible paths:

0 - based indexing is used.

- (0, 0) - (1, 1) - (2, 2) - (3, 3)
- (0, 0) - (1, 1) - (2, 0) - (3, 0)

For the first path, power left = 100 - 4 - 0 - 22 - 0 = 74.

For the second path, power left = 100 - 4 - 0 - 4 - 2 = 90.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     city[][] size n = 4
4        →     city[i] size m = 4
41 22 56 84
22 51 1 20
5 7 18 19
11 22 3 15
```

Sample Output

```
67
```

Explanation

Possible paths:

0 - based indexing is used.

- (0, 1) - (1, 0) - (2, 0) - (3, 0)
- (0, 1) - (1, 2) - (2, 1) - (3, 2)

For the first path, power left = 100 - 22 - 22 - 5 - 11 = 40.

For the second path, power left = 100 - 22 - 1 - 7 - 3 = 67.

Question - 57
Longest Execution Slot

You are analyzing CPU usage logs in a system where only one process runs at a time. There are 26 scheduled processes with IDs from 0 to 25, corresponding to characters 'a' through 'z'. The system starts running at time 0.

Only one process runs at a time, with the next beginning execution immediately after the previous one ends. Given the list of logs, where each entry is *[processNumber, endTime]* representing that the process with that ID ended at *endTime,* implement a function that determines which process had the longest single uninterrupted run. Consecutive runs of the same process are treated as separate.

The function *findLongestRun* will take the following input:

int runLogs[n][2]: an array of size n containing pairs of [processNumber, endTime]

The function should return the character ID of the process with the longest single run.

Example

runLogs = [[0, 3], [2, 5], [0, 9], [1, 15]]

In the example, the runs are as follows:

1. Process 0 runs from t = 0 to t = 3, i.e., for 3 units.
2. Process 2 runs from t = 3 to t = 5, i.e., for 2 units
3. Process 0 runs from t = 5 to t = 9, i.e., for 4 units
4. Process 1 runs from t = 9 to t = 15, i.e., for 6 units

The longest single run of a process is 6 units by process *1.*

Thus, the function should return 'b'

Constraints

- $1 \le n \le 10^5$
- $0 \le runLogs[i][0] \le 25$ ($0 \le i < n$)
- $1 \le runLogs[i][1] \le 10^8$ ($0 \le i < n$)
- Only one process has the longest execution run.
- *runLogs* is sorted in ascending order of *runLogs[i][1]*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *runLogs*.

The second line contains the integer *2,* the number of columns in each *runLogs[i]*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains two space-separated integers, *runLogs[i][0]* and *runLogs[i][1]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
3     →  runLogs[] size n = 3
2     →  runLogs[][] size = 2
0 2   →  runLogs = [[0, 2], [1, 3], [0, 7]]
1 3
0 7
```

Sample Output

```
a
```

Explanation

The time slots are as follows:

1. Process 0 runs for 2 - 0 = 2 units
2. Process 1 runs for 3 - 2 = 1 unit
3. Process 0 runs for 7 - 3 = 4 units

The longest execution is 4 units by process 0, ID = 'a'.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
5     →  runLogs[] size n = 5
2     →  runLogs[][] size = 2
0 1   →  runLogs = [[0, 1], [0, 3], [4, 5], [5, 6], [4, 10]]
0 3
4 5
5 6
4 10
```

Sample Output

```
e
```

Explanation

The time slots are as follows:

1. Process 0 runs for 1 - 0 = 1 unit
2. Process 0 runs for 3 - 1 = 2 units
3. Process 4 runs for 5 - 3 = 2 units
4. Process 5 runs for 6 - 5 = 1 unit
5. Process 4 runs for 10 - 6 = 4 units

The longest execution time is of 4 units by process 4 whose ID = 'e'. Note that the first two executions of process 0 are treated as separate.

Question - 58
Maximize Tool Utilization

You are managing *n* different developer tools. The number of uses of a tool of type *i* is given by *remainingUses[i]*.

The value a developer gains from using a tool is equal to its current remaining uses; more unused tools provide more benefit. However, each use decreases its remaining uses by 1.

Implement a function that finds the maximum total value achievable by assigning one tool use to each of the *m* developers, distributing the uses optimally across the available tools.

The function *maximizeToolValue* will take two inputs:
int remainingUses[n]: the remaining uses of each tool
int m: the number of developers.

The function should return a long integer, the maximum total value that can be obtained.

Example
remainingUses = [1,2,4]
m = 4

One of the optimal ways to allocate tool uses is as follows:

Number of allocations left	Tool Used (0-based indexing)	Total Value	Uses left
4	2	4	[1, 2, 3]
3	1	4 + 2 = 6	[1, 1, 3]
2	2	6 + 3 = 9	[1, 1, 2]
1	2	9 + 2 = 11	[1, 1, 1]

Hence, the maximum possible value is 11.

Constraints
- $1 \le n \le 10^5$
- $1 \le m \le 10^5$
- $1 \le remainingUses[i] \le 10^5$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *remainingUses*.
Each of the next *n* lines contains an integer *remainingUses[i]*.
The next line contains an integer *m*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
5        →    remainingUses[] size n = 5
10       →    remainingUses = [10, 10, 8, 9, 1]
10
8
9
1
6        →    m = 6
```

Sample Output 0

```
55
```

Explanation
One of the optimal ways to allocate tool uses is as follows:

Number of allocations left	Tool Used (0-based indexing)	Total Value	Uses left
6	0	10	[9, 10, 8, 9, 1]

5	0	10 + 9 = 19	[8, 10, 8, 9, 1]
4	0	19 + 8 = 27	[7, 10, 8, 9, 1]
3	1	27 + 10 = 37	[7, 9, 8, 9, 1]
2	1	37 + 9 = 46	[7, 8, 8, 9, 1]
1	3	46 + 9 = 55	[7, 8, 8, 8, 1]

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
4        →    remainingUses[] size n = 4
8        →    remainingUses = [8, 8, 8, 8]
8
8
8
4        →    m = 4
```

Sample Output 1

```
32
```

Explanation

The optimal way is to allocate one tool of each type. Hence, the maximum possible value = 8 + 8 + 8 + 8 = 32.

Question - 59
Token Frequency Analyzer

In a prompt engineering pipeline, a text string called *prompt* is tokenized into individual characters for analysis.

Implement a function that identifies the most frequently occurring token (character) in *prompt*. If multiple characters have the same maximum frequency, return the one that appears earliest in the string.

The function *getMostFrequentToken* takes the following input:
  string prompt: the text prompt consisting of ASCII characters ('a'-'z', 'A'-'Z', '0'-'9')

The function should return the most frequently occurring character (token), applying tie-breaking rules as specified.

Note: The analysis is case-sensitive. For example: 'A' and 'a' are different characters.

Example 1
prompt = "abcABCabcd"

Frequencies of each character: 'a': 2, 'b': 2, 'c': 2, 'd': 1, 'A': 1, 'B': 1, 'C': 1
Characters 'a', 'b', and 'c' occur twice, and 'a' appears first in *prompt*.
Hence, the answer is 'a'.

Constraints

- 10 ≤ length of *prompt* ≤ 10^4

▼ Input Format For Custom Testing

The first line contains a string, *prompt*, denoting the text to be analyzed.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           Function
-----           --------
helloworld  →   prompt = "helloworld"
```

Sample Output

```
l
```

Explanation

The character 'l' occurs *3* times, which is the highest frequency.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                                                      Function
-----                                                      --------
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz →   prompt =
"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz"
```

Sample Output

```
a
```

Explanation

All characters occur twice, and *'a'* occurs first.

Question - 60
Optimal File Decryption

In a file decryption system, two arrays named *scores* and *decryptionStatus* of size *n* are given, where:

- *scores[i]:* represents the sensitivity score of the i^{th} file
- *decryptionStatus[i]:* decryption status of i^{th} file, 1 if the i^{th} file is decrypted, 0 if encrypted

You are allowed to perform the following operation at most once:

- Choose a contiguous segment (subarray) of at most *k* files, and decrypt all files in that segment.

Implement a function that determines the maximum possible sum of scores of decrypted files after applying at most one such operation.

The function *maximizeDecryptionValue* takes the following inputs:
int scores[n]: the sensitivity scores of the files
int decryptionStatus[n]: the decryption status of files, 1 if the file is already decrypted, 0 otherwise
int k: maximum size of the subarray to decrypt in one operation

The function should return the maximum possible sum of the scores of decrypted files.

Example

n = 4
scores = [7, 4, 3, 5]
decryptionStatus = [1, 0, 0, 0]
k = 2

The first file with *scores[0]* = 7 is already decrypted. The objective is to choose a subarray of at most *k* files to decrypt.

There are three subarrays to choose from:

- [0, 1], *decryptionStatus'* = [1, 1, 0, 0], and the sum of scores is 7 + 4 = 11
- [1, 2], *decryptionStatus'* = [1, 1, 1, 0], and the sum of scores is 7 + 4 + 3 = 14
- [2, 3], *decryptionStatus'* = [1, 0, 1, 1], and the sum of scores is 7 + 3 + 5 = 15

The maximum sum is 15, so return 15.

Constraints

- $1 \le k \le n \le 10^5$
- $1 \le scores[i] \le 10^4$
- $0 \le decryptionStatus[i] \le 1$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in the array *scores*.
Each of the next *n* lines contains an integer, *scores[i]*.
The next line contains an integer, *n*, the number of elements in the array *decryptionStatus*.
Each of the next n lines contains an integer, *decryptionStatus[i]*.
The next line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
6        →     scores[] size n = 6
1        →     scores = [1 , 3 , 5 , 2 , 5 , 4]
3
5
2
5
4
6        →     decryptionStatus[] size n = 6
1        →     decryptionStatus = [1 , 1 , 0 , 1 , 0 , 0]
1
0
1
0
0
3        →     k = 3
```

Sample Output

```
16
```

Explanation

In this scenario, it is optimal to choose the subarray [2, 4] (0-based indexing) to get *decryptionStatus* = [1, 1, 1, 1, 1, 0]. The total score of decrypted files is 1 + 3 + 5 + 2 + 5 = 16.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
5        →     scores[] size n = 5
1        →     scores = [1 , 3 , 2 , 3 , 6]
3
2
3
6
5        →     decryptionStatus[] size n = 5
0        →     decryptionStatus = [0 , 0 , 0 , 1 , 0]
0
```

```
0
1
0
2        →     k = 2
```

Sample Output

```
9
```

Explanation

It is optimal to choose the subarray [4, 5] (0-based indexing) to get *decryptionStatus* = [0 , 0 , 0 , 1 , 1]. The total score of decrypted files is 3 + 6 = 9.

Question - 61
Retrieve Missing Label

In a machine learning pipeline, there is a given integer array called *labels*, where *labels[i]* denotes the class label of the i^{th} training data. Due to data corruption, some positive class labels might be missing from the dataset.

Implement a function that determines the k^{th} smallest positive label that is not present in the dataset.

The function *findMissingLabel* takes the following inputs:
int labels[n]: the existing class labels in the training data
long k: the rank of the missing positive label to identify (1-based index)

The function should return the k^{th} smallest positive label that is not present in the dataset.

Note: For reference, class labels are expected to start from 1 and increment by 1, i.e., the first valid label is 1, then 2, and so on.

Example
$n = 5$
labels = [1, 4, 7, 3, 4]
$k = 5$

The first five missing positive labels are [2, 5, 6, 8, 9]. The 5^{th} smallest missing label is 9, so the function should return 9.

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le labels[i] \le 10^9$
- $1 \le k \le 10^{12}$

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of elements in *labels*.
Each of the next *n* lines contains an integer, *labels[i]*.
The next line contains an integer *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     labels[] size n = 4
4        →     labels = [4, 2, 7, 5]
2
7
5
7        →     k = 7
```

Sample Output

```
11
```

Explanation

The first seven missing positive labels are [1, 3, 6, 8, 9, 10, 11]. The 7th smallest missing label is 11.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
6         →    labels[] size n = 6
2         →    labels = [2, 5, 7, 15, 5, 13]
5
7
15
5
13
6         →    k = 6
```

Sample Output

```
9
```

Explanation

The first six missing positive labels are [1, 3, 4, 6, 8, 9]. The 6th smallest missing label is 9.

Question - 62
API Gateway Logging

In a distributed web application with *m* gateway endpoints, *n* users trigger API requests. Each request may spawn multiple background jobs.

You are given two arrays *jobs* and *gateways* each of length *n* where:

- *jobs[i]:* the number of background jobs spawned from the *i*th request
- *gateways[i]:* the gateway ID that handles the *i*th request

Implement a function that determines the number of jobs handled by each gateway.

The function *computeGatewayLoad* takes the following inputs:
int jobs[n]: number of background jobs for each request
int gateways[n]: gateway ID for each request
int m: total number of gateway endpoints

The function should return an array of size *m*, where the value at the *i*th index is the number of jobs handled by the *i*th gateway.

Note: The question follows 1-based indexing of the arrays.

Example
jobs = [1, 2, 3]
gateways = [1, 1, 2]
m = 2

For three API requests across two gateway endpoints:

1. The first request triggers 1 background job along with itself → 1 + 1 = 2 jobs via Gateway 1.
2. The second request triggers 2 background jobs along with itself → 1 + 2 = 3 jobs via Gateway 1.
3. The third request triggers 3 background jobs along with itself → 1 + 3 = 4 jobs via Gateway 2.

Therefore, Gateway 1 handles a total of 5 jobs (2 + 3), and Gateway 2 handles 4 jobs.
Hence, the answer is [5, 4].

Constraints

- 1 ≤ *n, m* ≤ 100000
- 1 ≤ *jobs[i]* ≤ 1000
- 1 ≤ *gateways[i]* ≤ *m*

▼ Input Format for Custom Testing

The first line contains an integer, *n,* the size of the *jobs* array.
The following *n* lines contain an integer representing *jobs[i]*.
The following line contains an integer, *n*, the size of the *gateways* array.
The following *n* lines contain an integer representing *gateways[i]*.
The last line contains an integer, *m*, the number of gateways.

▼ Sample Case 0

Sample Input 0

```
STDIN   Function
-----   --------
3       jobs[] size n = 3
1       jobs = [1, 2, 3]
2
3
3       gateways[] size n = 3
1       gateways = [1, 2, 3]
2
3
3       m = 3
```

Sample Output 0

```
2
3
4
```

Explanation
For three API requests across three gateway endpoints:

1. The first request triggers 1 background job along with itself → 1 + 1 = 2 jobs via Gateway 1.
2. The second request triggers 2 background jobs along with itself → 1 + 2 = 3 jobs via Gateway 2.
3. The third request triggers 3 background jobs along with itself → 1 + 3 = 4 jobs via Gateway 3.

The totals for each gateway are:

- Gateway 1: 2 jobs
- Gateway 2: 3 jobs
- Gateway 3: 4 jobs

Hence, the answer is [2, 3, 4].

▼ Sample Case 1

Sample Input 1

```
STDIN   Function
-----   --------
3       jobs[] size n = 3
3       jobs = [3, 5, 1]
5
```

```
1
3       gateways[] size n = 3
1       gateways = [1, 1, 3]
1
3
3       m = 3
```

Sample Output 1

```
10
0
2
```

Explanation

For three API requests across three gateway endpoints:

- The first request triggers 3 background jobs along with itself → 1 + 3 = 4 jobs via Gateway 1.
- The second request triggers 5 background jobs along with itself → 1 + 5 = 6 jobs via Gateway 1.
- The third request triggers 1 background job along with itself → 1 + 1 = 2 jobs via Gateway 3.

The totals for each gateway are:

- Gateway 1: 4 + 6 = 10 jobs
- Gateway 2: 0 jobs
- Gateway 3: 2 jobs

Hence, the answer is [10, 0, 2].

Question - 63

Feature Deployment Time

You are managing rollouts for a platform with *n* features, numbered 1 to *n*. Each feature has:

- a setup phase taking *setupTime[i]* minutes (first deployment only)
- a testing phase taking *testTime[i]* minutes (every deployment)

When a feature is deployed for the first time, both the setup and testing phases must be completed. However, only the testing phase is required on redeployments of the same feature, e.g., for bug fixes or enhancements.

Deployment is sequential:

- Only Feature 1 is initially deployable
- Feature *i* + 1 can be deployed only after deploying Feature *i* at least once

Implement a function to compute the minimum total time to perform exactly *totalDeploys* feature deployments (initial or repeat).

The function *getMinTime* will take three inputs:
  int setupTime[n]: setup time for each feature
  int testTime[n]: testing time for each feature
  int totalDeploys: total number of feature deployments (initial + repeat)

The function should return a long integer denoting the minimum total time required to perform exactly *totalDeploys* feature deployments.

Example

n = 3
totalDeploys = 4
setupTime = [3, 2, 4]
testTime = [4, 3, 2]

The optimal way to complete four feature deployments is:

- Deploy Feature 1 for the first time: This takes *setupTime[0]* + *testTime[0]* = 7 minutes (setup + testing).
- Deploy Feature 2 for the first time: This takes *setupTime[1]* + *testTime[1]* = 5 minutes.
- Redeploy Feature 2: This takes *testTime[1]* = 3 minutes.
- Redeploy Feature 2: This takes *testTime[1]* = 3 minutes.

Hence, the minimum total time is 18 minutes.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le$ *setupTime[i], testTime[i]* $\le 2 * 10^5$
- $1 \le$ *totalDeploys* $\le 2 * 10^5$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of *setupTime*.
Each of the next *n* lines contains an integer *setupTime[i]*.
The next line contains an integer *n*, the size of *testTime*.
Each of the next *n* lines contains an integer *testTime[i]*.
The next line contains an integer *totalDeploys*.

▼ Sample Case 0

Sample Input 0

```
STDIN      Function
-----      --------
2      →   setupTime[] size n = 2
1      →   setupTime = [1, 2]
2
2      →   testTime[] size n = 2
3      →   testTime = [3, 4]
4
1      →   totalDeploys = 1
```

Sample Output 0

```
4
```

Explanation

Deploying Feature 1 for the first time: This takes *setupTime[0]* + *testTime[0]* = 4 minutes.

▼ Sample Case 1

Sample Input 1

```
STDIN      Function
-----      --------
3      →   setupTime[] size n = 3
1      →   setupTime = [1, 1, 1]
1
1
3      →   testTime[] size n = 3
2      →   testTime = [2, 3, 4]
3
4
3      →   totalDeploys = 3
```

Sample Output 1

```
7
```

Explanation

The optimal way to complete three feature deployments is:

- Deploy Feature 1 for the first time: This takes *setupTime[0]* + *testTime[0]* = 3 minutes (setup + testing).
- Redeploy Feature 1: This takes *testTime[0]* = 2 minutes.

- Redeploy Feature 1 again: This takes another *testTime[0]* = 2 minutes.

Hence, the minimum total time is 7 minutes.

Question - 64
Maximize The Equality

Implement a function that finds the maximum number of pairs of consecutive equal characters that can be formed in any subsequence of a given binary string *s*.

The function *getMaxPairs* will take one input:
  string s: the string

The function should return an integer representing the maximum number of pairs of consecutive equal characters that can be formed.

Note: A subsequence of a string can be derived from the string by removing zero or more characters without changing the order of the remaining characters.

Example
s = "10110"

For *s* = "10110", the maximum pairs of consecutive equal characters are formed by choosing all '1's to create the subsequence "111", resulting in 2 pairs (0, 1) and (1, 2) (0-based indexing).

Hence, the maximum number of pairs is 2.

Constraints

- 1 ≤ length of *s* ≤ 2 * 10^5
- It is guaranteed that string *s* contains only '0' and '1'.

▼ Input Format for Custom Testing

The first line contains the string *s*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
00011     →    s = "00011"
```

Sample Output 0

```
3
```

Explanation
The maximum pairs of consecutive equal characters are formed by choosing whole *s* to create the subsequence "00011", resulting in 3 pairs (0, 1), (1, 2), and (3, 4) (0-based indexing).

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
101010101 →    s = "101010101"
```

Sample Output 1

```
4
```

Explanation

The maximum pairs of consecutive equal characters is formed by choosing all '1's to create the subsequence "11111", resulting in 4 pairs (0, 1), (1, 2), (2, 3), and (3, 4) (0-based indexing).

Question - 65

Task Completion Order

You have a queue of *n* tasks numbered from 1 to *n*, where each task requires *time[i]* units to complete. You will process tasks in order from 1 to *n*. In one turn, you can process at most *m* units of time for a task.

If a task needs more than *m* units, it is moved to the back of the queue with its remaining time. A task is removed from the queue once it is fully completed.

Implement a function that finds the order in which tasks are completed.

The function *getTaskOrder* will take two inputs:
int time[n]: the time required for each task to complete
int m: the maximum units of time for any task you can process in one turn

The function should return an array of integers representing the order in which tasks are completed.

Example

n = 3
m = 1
time = [1, 2, 3]

Simulation Steps

Step	Queue before (1-based indexing)	Task Processed	Time Processed	Queue after (1-based indexing)	Completed Tasks(1-based indexing)
1	[1, 2, 3]	1	1	[2, 3]	[1]
2	[2, 3]	2	1	[3, 2]	[1]
3	[3, 2]	3	1	[2, 3]	[1]
4	[2, 3]	2	1	[3]	[1, 2]
5	[3]	3	1	[3]	[1, 2]
6	[3]	3	1	[]	[1, 2, 3]

So, the final order of completion is [1, 2, 3].

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le m, time[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *time*.
Each of the next *n* lines contains an integer *time[i]*.
The first line contains an integer *m*, the maximum units of time for any task you can process in one turn.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
4         →    time size[] n = 4
2         →    time = [2, 1, 4, 3]
1
4
3
3         →    m = 3
```

Sample Output 0

```
1
2
4
3
```

Explanation

Simulation Steps

Step	Queue before (1-based indexing)	Task Processed	Time Processed	Queue after (1-based indexing)	Completed Tasks (1-based indexing)
1	[1, 2, 3, 4]	1	2	[2, 3, 4]	[1]
2	[2, 3, 4]	2	1	[3, 4]	[1, 2]
3	[3, 4]	3	3	[4, 3]	[1, 2]
4	[4, 3]	4	3	[3]	[1, 2, 4]
5	[3]	3	1	[]	[1, 2, 4, 3]

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
5         →    time size[] n = 5
4         →    time = [4, 4, 5, 3, 5]
4
5
3
5
3         →    m = 3
```

Sample Output 1

```
4
1
2
3
5
```

Explanation

Simulation Steps

Step	Queue before (1-based indexing)	Task Processed	Time Processed	Queue after (1-based indexing)	Completed Tasks (1-based indexing)
1	[1, 2, 3, 4, 5]	1	3	[2, 3, 4, 5, 1]	[]
2	[2, 3, 4, 5, 1]	2	3	[3, 4, 5, 1, 2]	[]

3	[3, 4, 5, 1, 2]	3	3	[4, 5, 1, 2, 3]	[]
4	[4, 5, 1, 2, 3]	4	3	[5, 1, 2, 3]	[4]
5	[5, 1, 2, 3]	5	3	[1, 2, 3, 5]	[4]
6	[1, 2, 3, 5]	1	1	[2, 3, 5]	[4, 1]
7	[2, 3, 5]	2	1	[3, 5]	[4, 1, 2]
8	[3, 5]	3	2	[5]	[4, 1, 2, 3]
9	[5]	5	2	[]	[4, 1, 2, 3, 5]

Question - 66
Cost-Efficient Task Cleanup

You are given an array *taskCosts* of size *n*, where each element represents the cost of executing a task.
For each task, you can either:

- Retain it, paying its individual cost, or
- Remove it using a cleanup operation.

Cleanup operations are available in bundles, each with the following properties:

- It provides *operationsPerBundle* cleanup operations.
- It costs *bundleCost* per bundle.
- Multiple bundles can be allocated.
- Unused operations within a bundle are allowed.

Implement a function to compute the minimum total cost, defined as the sum of execution costs of all retained tasks plus the total cost of allocated cleanup bundles.

The function *getMinimumCost* will take three inputs:
int taskCosts[n]: an array representing the execution cost of each task.
int operationsPerBundle: the number of cleanup operations available in one bundle.
int bundleCost: the cost to allocate one cleanup bundle.

The function should return a long integer denoting the minimum total system cost, which includes the cost of retained tasks and the cost of allocated cleanup bundles.

Example
n = 5
taskCosts = [7, 1, 6, 3, 6]
operationsPerBundle = 2
bundleCost = 10

An optimal way to minimize cost

Task number (0-based)	Retain Cost	Use Cleanup
0	7	yes
1	1	no
2	6	yes

3	3	no
4	6	no

There are two operations to clean up, so one cleanup bundle is required.
Hence, the total cost is 1 * 10 + (0 + 1 + 0 + 3 + 6) = 20.

Constraints

- 1 ≤ *operationsPerBundle* ≤ *n* ≤ 2 * 10^5
- 1 ≤ *bundleCost, taskCosts[i]* ≤ 2 * 10^5

▼ Input Format for Custom Testing

The first line contains an integer *n*, representing the size of the array *taskCosts*.
The next *n* lines contain an integer *taskCosts[i]*.
The next line contains an integer *operationsPerBundle*.
The next line contains an integer *bundleCost*.

▼ Sample Case 0

Sample Input 0

```
STDIN      Function
-----      --------
3      →   taskCosts[] n = 3
1      →   taskCosts = [1, 2, 3]
2
3
1      →   operationsPerBundle = 1
10     →   bundleCost = 10
```

Sample Output 0

```
6
```

Explanation

An optimal way to minimize cost

Task number (0-based)	Retain Cost	Use Cleanup
0	1	no
1	2	no
2	3	no

The total operations used is 0. So, the number of bundles required is 0.
Hence, the total cost is 0 * 1 + (1 + 2 + 3) = 6.

▼ Sample Case 1

Sample Input 1

```
STDIN      Function
-----      --------
4      →   taskCosts[] n = 4
3      →   taskCosts = [3, 2, 3, 1]
2
3
1
4      →   operationsPerBundle = 4
2      →   bundleCost = 2
```

Sample Output 1

```
2
```

Explanation

An optimal way to minimize cost

Task number (0-based)	Retain Cost	Use Cleanup
1	3	yes
2	2	yes
3	3	yes
4	1	yes

The total operations used is 4. So, the number of batches required is 1.
Hence, the total cost is 1 * 2 + (0 + 0 + 0 + 0) = 2.

Question - 67
Array Range Minimization

Implement a function that determines the minimum possible difference between the maximum and minimum elements of an array *arr1* of size *n*. Any number of swap operations can be performed with a second array, *arr2,* of the same size.

A swap operation consists of:

- Selecting an index *i* (0 ≤ *i* < *n*) and swapping the elements *arr1[i]* and *arr2[i]*.

The function *getMinimumDifference* will take two inputs:
int arr1[n]: the first array.
int arr2[n]: the second array.

The function should return an integer denoting the minimum possible value of max(*arr1*) - min(*arr1*) after performing the swap operations optimally.

Example
n = 3
arr1 = [4, 6, 1]
arr2 = [3, 2, 5]

Here, the underlined elements represent the elements to be swapped and their modified values. The indices in the first column use 0-based indexing.

An optimal sequence of operations

Chosen Index	*arr1* before	*arr1* after	max(*arr1*) - min(*arr1*)
2	[4, 6, 1]	[4, 6, 5]	6 - 4 = 2
0	[4, 6, 5]	[4, 2, 1]	4 - 1 = 3
1	[4, 6, 1]	[3, 2, 1]	3 - 1 = 2

Hence, the optimal answer is 2.

Constraints

- $1 \le n \le 10^5$
- $1 \le arr1[i], arr2[i] \le 10^9$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of *arr1*.
Each of the next *n* lines contains an integer *arr1[i]*.
The next line contains an integer *n*, the size of *arr2*.
Each of the next *n* lines contains an integer *arr2[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN                   Function
-----                   --------
3            →          arr1[] size n = 3
5            →          arr1 = [5, 1, 3]
1
3
3            →          arr2[] size n = 3
3            →          arr2 = [3, 2, 1]
2
1
```

Sample Output 0

```
1
```

Explanation

An optimal sequence of operations

Chosen Index	*arr1* before	*arr1* after	max(*arr1*) - min(*arr1*)
0	[5, 1, 3]	[3, 1, 3]	3 - 1 = 2
1	[3, 1, 3]	[3, 2, 3]	3 - 2 = 1

▾ Sample Case 1

Sample Input 1

```
 STDIN                  Function
-----                   --------
4            →          arr1[] size n = 4
1            →          arr1 = [1, 4, 5, 2]
4
5
2
4            →          arr2[] size n = 4
1            →          arr2 = [1, 5, 4, 3]
5
4
3
```

Sample Output 1

```
3
```

Explanation

An optimal sequence of operations

Chosen Index	*arr1* before	*arr1* after	max(*arr1*) - min(*arr1*)

2	[1, 4, 5, 2]	[1, 4, 4, 2]	4 - 1 = 3

Question - 68
Binary String Reordering

Implement a function that determines the minimum number of operations needed to transform a binary string *s* into a target binary string *t,* both of length *n*.

In one operation, you can do the following:

- Choose a pair of integers (*i*, *j*) (where 0 ≤ *i* < *j* ≤ *n* - 1) and swap *s[i]* and *s[j]*, provided one of the following conditions holds:
  - *s[i]* = 0 and *s[i+1]* = ⋯ = *s[j]* = 1 (i.e., *s[i]* = 0 and all elements from *s[i+1]* through *s[j]* = 1).
  - *s[i]* = ⋯ = *s[j−1]* = 1 and *s[j]* = 0 (i.e., all elements from *s[i]* through *s[j−1]* = 1 and *s[j]* = 0).

The function *getMinOperations* will take two inputs:
string s: the source string.
string t: the target string.

The function should return an integer denoting the minimum number of operations needed, or -1 if it is impossible to make *s* equal to *t*.

Example
n = 7
s = "1110110"
t = "1010111"

An optimal sequence of operations

s before	Chosen indices (*i*, *j*) (0-based)	*s* after
1110110	(1, 3)	1011110
1011110	(3, 6)	1010111

It requires at least two operations to make *s* equal to *t;* hence, the answer is 2.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- It is guaranteed that strings *s* and *t* contain only '0's and '1's.

▼ Input Format for Custom Testing

The first line contains a string *s*.
The next line contains a string *t*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
0111101   →    s = "0111101"
0011111   →    t = "0011111"
```

Sample Output 0

```
1
```

Explanation

An optimal sequence of operations

s before	Chosen indices (*i*, *j*) (0-based)	*s* after
0111101	(1, 5)	0011111

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
111100   →     s = "111100"
111000   →     t = "111000"
```

Sample Output 1

```
-1
```

Explanation

It is impossible to make *s* equal to *t*.

Question - 69
Binary Classification

In a binary classification system, each data point has a numerical score and a binary label (0 or 1). The system predicts:

- Label 0 if the score is less than a threshold X.
- Label 1 if the score is greater than or equal to X.

Implement a function to find the maximum number of correctly classified data points by choosing an optimal threshold X.

The function *getOptimalClassification* will take two inputs:
int scores[n]: the scores of the data points
string labels: a binary string of length *n* representing the actual labels of the data points

The function should return an integer denoting the maximum number of correctly classified data points using some threshold X.

Example
scores = [60, 50, 50, 50, 60]
labels = "10101"

X	Predicted labels	Explanation	*labels*	Correct Classifications
40	"11111"	All scores are ≥ 40.	"10101"	3
50	"11111"	All scores are ≥ 50.	"10101"	3
55	"10001"	Points 1 and 5 (1-based indexing) have scores ≥ 55.	"10101"	4
60	"10001"	Points 1 and 5 (1-based indexing) have scores ≥ 60.	"10101"	4
65	"00000"	No point has a score ≥ 65.	"10101"	2

There is no value of *X* such that all predicted labels are correct. Hence, the answer is 4.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le scores[i] \le 10^9$
- It is guaranteed that the string *labels* contains the characters '0' and '1' only.

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *scores*.
Each of the next *n* lines contains an integer, *scores[i]*.
The next line contains the string *labels*.

▾ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
5         →    scores[] size n = 5
60        →    scores = [60, 45, 30, 40, 80]
45
30
40
80
10101     →    labels = "10101"
```

Sample Output 0

```
4
```

Explanation

X	Predicted labels	Explanation	*labels*	Correct Classifications
30	"11111"	All scores are ≥ 30.	"10101"	3
40	"11011"	Points 1, 2, 4, and 5 (1-based indexing) have scores ≥ 40.	"10101"	2
45	"11001"	Points 1, 2, and 5 (1-based indexing) have scores ≥ 45.	"10101"	3
60	"10001"	Points 1 and 5 (1-based indexing) have scores ≥ 60.	"10101"	4
80	"00001"	Only point 5 has a score ≥ 80.	"10101"	3

There is no value of *X* such that all predicted labels are correct. Hence, the answer is 4.

▾ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
3         →    scores[] size n = 3
1         →    scores = [1, 2, 3]
2
3
011       →    labels = "011"
```

Sample Output 1

```
3
```

Explanation

X	Predicted labels	Explanation	*labels*	Correct Classifications
0	"111"	The score of all points is ≥ 0.	"011"	2
1	"111"	The score of all points is ≥ 1.	"011"	2
2	"011"	Points 2 and 3 (1-based indexing) have score ≥ 1.	"011"	3
3	"001"	Only point 3 (1-based indexing) has score ≥ 2.	"011"	2

Choosing *X* = 2 gives the correct predicted label for all data points. Hence, the answer is 3.

Question - 70
Optimal Subarray Removal

Implement a function that finds the maximum length of a subarray that can be removed from the array *arr* of length *n* such that the difference between the size of the remaining array and its number of distinct elements is maximized.

The function *getMaxDifference* will take one input:
int arr[n]: the elements of the array *arr*.

The function should return an integer representing the maximum length of a subarray that can be removed from the array.

Note: A subarray is a contiguous segment of an array after deleting several (possibly zero) elements from the start and/or end of the array.

Example
n = 4
arr = [2, 1, 3, 2]

An optimal subarray is [1, 3], after which *arr* becomes [2, 2]. This has length 2 and 1 distinct element, so the required difference is 1.
It can be shown that the maximum possible difference is 1, and the maximum length of a subarray that can be removed for this is of length 2.
Hence, the answer is 2.

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le arr[i] \le n$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
5      →    arr[] size n = 5
3      →    arr = [3, 3, 2, 3, 5]
3
2
3
5
```

Sample Output 0

```
1
```

Explanation

An optimal subarray is [2]. After it is removed, *arr* is [3, 3, 3, 5]. This has length 4 and 2 distinct elements, and their difference is 2.
Hence, the maximum length of the subarray is 1.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
-----       --------
6      →    arr[] size n = 6
3      →    arr = [3, 6, 6, 4, 5, 2]
6
6
4
5
2
```

Sample Output 1

```
3
```

Explanation

An optimal subarray is [4, 5, 2]. After it is removed, *arr* is [3, 6, 6]. This has length 3 and 2 distinct elements, and their difference is 1.

Question - 71
Largest Sum

Implement a function that finds the maximum possible sum of an array *arr* of size *n* after performing an operation any number of times (including zero).

In an operation:

- Choose any two indices *i* and *j* such that $0 \le i, j < n$, and $i \ne j$, and set *arr[i]* to *−arr[i]* and *arr[j]* to *−arr[j]*.

The function *getMaxSum* will take the following input:
  int arr[n]: the given array

The function should return a long integer denoting the maximum possible sum of the array *arr* after performing the operation any number of times.

Example

n = 3

arr = [-5, -1, 7]

Consider the following operation (0-based indexing):

- Choose *i* = 0 and *j = 1*: the array becomes [5, 1, 7], and the sum becomes 13.

13 is the maximum possible sum. Hence, return 13.

Constraints

- $2 \le n \le 2 * 10^5$
- $-10^9 \le arr[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the length of the array *arr*.
Each of the next *n* lines contains an integer, *arr[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN           FUNCTION
----            --------
5          →    arr[] size n = 5
-2         →    arr = [-2, 1, -2, -3, -3]
1
-2
-3
-3
```

Sample Output 0

```
11
```

Explanation

Consider the following sequence of operations (0-based indexing):

- Choose *i* = 0 and *j* = 2: the array becomes [2, 1, 2, -3, -3], and the sum becomes -1.
- Choose *i* = 3 and *j* = 4: the array becomes [2, 1, 2, 3, 3], and the sum becomes 11.

▾ Sample Case 1

Sample Input 1

```
STDIN           FUNCTION
----            --------
3          →    arr[] size n = 3
1          →    arr = [1, -3, 4]
-3
4
```

Sample Output 1

```
6
```

Explanation

Consider the following operation (0-based indexing):

- Choose *i* = 1 and *j* = o: the array becomes [–1, 3, 4], and the sum becomes 6.

Question - 72
Maximize Consecutive Ones

Implement a function that determines the maximum number of consecutive '1's that can be achieved in a given binary string *s* of length *n* after performing an operation on at most one substring.

An operation consists of :

- Selecting a substring of length *k* and changing all its '0's to '1's.

The function *getMaximumConsecutiveOnes* will take two inputs:

string s: the given binary string

int k: the length of the substring that can be modified

The function should return an integer, denoting the maximum number of consecutive '1's that can be achieved after at most one operation.

Example

s = "01001"

k = 2

Here, the underlined substring of length *k* represents the one to be modified.

All possible operations

s before	*s* after	Maximum number of consecutive '1's
"01001"	"11001"	2
"01001"	"01101"	2
"01001"	"01111"	4
"01001"	"01011"	2

Hence, the answer is 4.

Constraints

- $1 \le k \le n \le 2*10^5$
- It is guaranteed that *s* consists of characters '0' and '1' only.

▾ Input Format for Custom Testing

The first line contains a binary string *s*.

The next line contains an integer *k*, the length of the substring that can be modified.

▾ Sample Case 0

Sample Input 0

```
STDIN                    Function
-----                    --------
0101            →        s = "0101"
2               →        k = 2
```

Sample Output 0

```
3
```

Explanation

Here, the underlined substring of length *k* represents the one to be modified.

All possible operations

s before	*s* after	Maximum number of consecutive '1's
"0101"	"1101"	2
"0101"	"0111"	3
"0101"	"0111"	3

▾ Sample Case 1

Sample Input 1

```
STDIN                       Function
-----                       --------
```

```
100001          →        s = "100001"
3               →        k = 3
```

Sample Output 1

```
4
```

Explanation
Here, the underlined substring of length *k* represents the one to be modified.

All possible operations

s before	*s* after	Maximum number of consecutive '1's
"100001"	"111001"	3
"100001"	"111101"	4
"100001"	"101111"	4
"100001"	"100111"	3

Question - 73
Optimal Subarray Deletion

Implement a function that determines the maximum length of a subarray that can be removed from a given permutation array *arr* of length *n* such that the remaining array remains a non-empty permutation.

The function *getMaximumSubarray* will take one input:
int arr[n]: the elements of *arr*.

The function should return an integer denoting the maximum length of a subarray that can be removed.

Note:

- A subarray of an array is a contiguous segment of the array.
- A permutation of length *n* is a sequence consisting of all integers from 1 to *n* exactly once in any order.

Example
n = 3
arr = [3, 1, 2]

Some possible subarray removals

Range of deletion(0-based indexing)	*arr* before	*arr* after	Is this a permutation?
[0, 1]	[3, 1, 2]	[2]	No
[0, 0]	[3, 1, 2]	[1, 2]	Yes
[1, 2]	[3, 1, 2]	[3]	No

The underlined elements represent the subarray that is removed.

Hence, the maximum length of a subarray that can be removed is 1.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le arr[i] \le n$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN               Function
-----               --------
4           →       arr[] size n = 4
3           →       arr = [3, 2, 4, 1]
2
4
1
```

Sample Output 0

```
3
```

Explanation

Some possible subarray removals

Range of deletion(0-based indexing)	*arr* before	*arr* after	Is this a permutation?
[1, 2]	[3, 2, 4, 1]	[3, 1]	No
[2, 3]	[3, 2, 4, 1]	[3, 2]	No
[0, 2]	[3, 2, 4, 1]	[1]	Yes
[2, 2]	[3, 2, 4, 1]	[3, 2, 1]	Yes

The underlined elements represent the subarray that is removed.

Hence, the maximum length of a subarray that can be removed is 3.

▾ Sample Case 1

Sample Input 1

```
STDIN               Function
-----               --------
5           →       arr[] size n = 5
3           →       arr = [3, 1, 5, 4, 2]
1
5
4
2
```

Sample Output 1

```
2
```

Explanation

Some possible subarray removals

Range of deletion(0-based indexing)	*arr* before	*arr* after	Is this a permutation?
[1, 3]	[3, 1, 5, 4, 2]	[3, 2]	No

[2, 2]	[3, 1, 5, 4, 2]	[3, 1, 4, 2]	Yes
[0, 3]	[3, 1, 5, 4, 2]	[2]	No
[2, 3]	[3, 1, 5, 4, 2]	[3, 1, 2]	Yes
[3, 4]	[3, 1, 5, 4, 2]	[3, 1, 5]	No

The underlined elements represent the subarray that is removed.

Hence, the maximum length of a subarray that can be removed is 2.

Question - 74
Subsequence Update

You are given an integer array *arr* of length *n* and two integers *l* and *r* representing the segment [*l*, *r*] (0-based) (where $0 \le l \le r < n$).

Implement a function that finds the minimum possible value of the sum *arr[l] + arr[l+1] + ... + arr[r]* after reversing exactly one subsequence of *arr*.

The function *getMinimumSum* will take three inputs:
int arr[n]: the elements of *arr*.
int l: the starting index of the segment.
int r: the ending index of the segment.

The function should return a long integer representing the minimum possible value of the sum of the segment.

Note: A sequence *b* is a subsequence of a sequence *a* if *b* can be obtained from *a* by the deletion of several (possibly, zero or all) elements from arbitrary positions.

Example
arr = [1, 2, 3]
l = 1
r = 2

Reversing Subsequences

Subsequence Reversed (0-based indexing)	Resulting Array	Sum of Segment
{0, 1}	[2, 1, 3]	1 + 3 = 4
{0, 2}	[3, 2, 1]	2 + 1 = 3
{1, 2}	[1, 3, 2]	3 + 2 = 5

The minimum value of the sum of the segments is 3.

Constraints
- $1 \le n \le 2 * 10^5$
- $0 \le l \le r < n$
- $1 \le arr[i] \le 10^8$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr*.

Each of the next *n* lines contains an integer *arr[i]*.

The next line contains an integer *l*, the starting index of the segment.

The next line contains an integer *r*, the ending index of the segment.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
5         →    arr[] size n = 5
3         →    arr = [3, 3, 2, 3, 5]
3
2
3
5
1         →    l = 1
4         →    r = 4
```

Sample Output 0

```
11
```

Explanation

Reversing Subsequences

Subsequence Reversed (0-based indexing)	Resulting Array	Sum of Segment
{1, 2}	[3, 2, 3, 3, 5]	3 + 2 + 3 + 5 = 13
{1, 3}	[3, 3, 2, 3, 5]	3 + 2 + 3 + 5 = 13
{0, 4}	[5, 3, 2, 3, 3]	3 + 2 + 3 + 3 = 11

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
5         →    arr[] size n = 6
3         →    arr = [3, 6, 6, 4, 3, 2]
6
6
4
3
2
0         →    l = 0
2         →    r = 2
```

Sample Output 1

```
8
```

Explanation

Reversing Subsequences

Subsequence Reversed (0-based indexing)	Resulting Array	Sum of Segment
{1, 2}	[3, 6, 6, 4, 3, 2]	15
{1, 3}	[3, 4, 6, 6, 3, 2]	13
{1, 2, 4, 5}	[3, 2, 3, 4, 6, 6]	8

Question - 75
Sum Pairs

Implement a function that counts the number of ordered pairs of 0-based indices *(x, y)* in an array *arr* of size *n* such that the sum *arr[x] + arr[y]* is greater than or equal to *lowerLimit* and less than or equal to *upperLimit*.

The function *getNumberOfPairs* will take three inputs:

int arr[n]: The given array
int lowerLimit: The lower limit of the sum
int upperLimit: The upper limit of the sum

The function should return a long integer, the number of pairs of indices *(x, y)* such that their sum *(arr[x] + arr[y])* is between *lowerLimit* and *upperLimit*, inclusive.

Note: An ordered pair is one in which the order of elements matters, that is, *(a,b)* is equal to *(u,v)* if and only if *a=u* and *b=v*.

Example
n = 4
arr = [4, 6, 8, 15]
lowerLimit = 18
upperLimit = 28

The pairs of 0-based indices with sum greater than or equal to 18 and less than or equal to 28 are (0, 3), (3, 0), (1, 3), (3, 1), (3, 2), and (2, 3).
Thus, the answer is 6.

Constraints

- $1 \le n \le 2*10^5$
- $-10^9 \le arr[i] \le 10^9$
- $-10^9 \le lowerLimit \le upperLimit \le 10^9$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array arr.
Each of the next *n* lines contains an integer arr*[i]*.
The next line contains an integer *lowerLimit*.
The next line contains an integer *upperLimit*.

▾ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
4        →    arr[] size n = 4
5        →    arr = [5, 10, 15, 25]
10
15
25
21       →    lowerLimit = 21
29       →    upperLimit = 29
```

Sample Output 0

```
2
```

Explanation
The pairs of indices (0-based indexing) with a sum between 21 and 29 are (1, 2) and (2,1).

▾ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
4        →    arr[] size n = 4
8        →    arr = [8, 2, 6, 3]
2
6
3
6        →    lowerLimit = 6
12       →    upperLimit = 12
```

Sample Output 1

```
10
```

Explanation

All the ordered pairs of indices (0-based indexing) with sums between 6 and 12, inclusive, are: (0, 1), (0, 3), (1, 0), (1, 2), (2, 1), (2, 2), (2, 3), (3, 0), (3, 2) and (3, 3).

Question - 76
Shift Sort

Implement a function that determines the minimum number of operations required to sort an array *arr* of size *n* in non-decreasing order using the following operations any number of times (possibly zero):

- Extract the first element of the array and insert it at the end.
- Then swap that element with the previous one until it becomes the first or is strictly greater than the previous one.

The function *minOperationsToSort* will take the following input:
int arr[n]: The elements of the array *arr*.

The function should return an integer denoting the minimum number of operations needed to sort the array in non-decreasing order, or -1 if sorting is impossible.

Example
arr = [5, 3, 1]

Operations on the array

Array before the operation	Array after the operation
[5, 3, 1]	[3, 1, 5]
[3, 1, 5]	[1, 3, 5]

The array *arr* can be sorted in non-decreasing order by applying 2 operations. Hence, the answer is 2.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le arr[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains the integer *n,* the size of array *arr*.
The next *n* lines contain an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
```

```
5          →      arr[] size n = 5
6          →      arr = [6, 4, 1, 2, 5]
4
1
2
5
```

Sample Output 0

```
2
```

Explanation

Operations on the array

Array before the operation	Array after the operation
[6, 4, 1, 2, 5]	[4, 1, 2, 5, 6]
[4, 1, 2, 5, 6]	[1, 2, 4, 5, 6]

The array *arr* can be sorted in non-decreasing order by applying 2 operations. Hence, the answer is 2.

▼ Sample Case 1

Sample Input 1

```
STDIN              Function
-----              --------
5          →       arr[] size n = 5
1          →       arr = [1, 2, 4, 2, 2]
2
4
2
2
```

Sample Output 1

```
-1
```

Explanation
The array *arr* cannot be sorted in non-decreasing order by applying any number of operations. Hence, the answer is -1.

Question - 77
Smallest Valid Subarray

You are given an array *arr* of size *n,* which contains all the numbers from 1 to *n* exactly once. A subarray is called 'valid' if, after selecting exactly *k* elements from it, we can rearrange them to form a set of *k* consecutive integers.

Implement a function that finds the size of the smallest valid subarray in an array *arr*.

The function *getSmallestValidSubarray* will take two inputs:
int arr[n]: The array from which the subarray is to be selected.
int k: the number of consecutive integers required in the subarray.

The function should return an integer representing the size of the smallest valid subarray in *arr*.

Note: A subarray is a contiguous segment of an array that can be obtained by deleting several (preferably zero) elements from the start and end of the array, without reordering or skipping any elements in between.

Example
arr = [10, 1, 6, 8, 7, 2, 5, 9, 3, 4]

k = 5

Consider the subarray [6, 8, 7, 2, 5, 9]. We can select five consecutive numbers (5, 6, 7, 8, 9). There is no smaller valid subarray of *arr*.
Hence, the answer is 6.

Constraints

- $1 \le k \le n \le 2 * 10^5$
- $1 \le arr[i] \le n$
- All elements in *arr* are distinct

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.
The next line contains an integer *k*.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
4       →      arr[]  size n = 4
2       →      arr = [2, 3, 1, 4]
3
1
4
2       →      k = 2
```

Sample Output 0

```
2
```

Explanation
Conider the valid subarray [2, 3]. We can select two consecutive numbers (2, 3).
Hence, the answer is 2.

▼ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
7       →      arr[]  size n = 7
3       →      arr = [3, 7, 2, 5, 1, 4, 6]
7
2
5
1
4
6
6       →      k = 6
```

Sample Output 1

```
7
```

Explanation
The only valid subarray of *arr* is *arr* itself, i.e., [3, 7, 2, 5, 1, 4, 6]. We can select six consecutive numbers (1, 2, 3, 4, 5, 6) or (2, 3, 4, 5, 6, 7).
Hence, the answer is 7.

Question - 78
Largest String

Implement a function that removes the minimum number of characters from a given string *s,* such that in the remaining string, *sRem*, these two conditions are satisfied:

- The length of *sRem* is even.
- The characters *sRem[i]* and *sRem[i+1]* are equal for all even *i (0 ≤ i < length(sRem) - 1)*.

The function *getMinimumMoves* will take one input:
string s: The given string

The function should return an integer denoting the minimum number of characters removed to make the given string satisfy the required conditions.

Example
s = "abb"

If we remove the first character, i.e., 'a', the string becomes "bb", which satisfies both conditions.
Thus, the minimum number of characters to remove is 1.

Constraints

- 1 ≤ *length(s)* ≤ 2 * 10^5

▾ Input Format for Custom Testing

The first line contains a string *s*.

▾ Sample Case 0

Sample Input 0

```
STDIN             Function
------            --------
bbcbb      →      s = "bbcbb"
```

Sample Output 0

```
1
```

Explanation
The original string is of odd length. Thus, we need to remove at least one character. If we remove the only character 'c', the remaining string will satisfy both conditions.

▾ Sample Case 1

Sample Input 1

```
STDIN             Function
------            --------
kkkj       →      s = "kkkj"
```

Sample Output 1

```
2
```

Explanation
The current string does not satisfy the criteria. There is only one 'j', which must be removed. Now, the length of the string is odd, so we must remove another character, 'k'. After removing one 'k', the string satisfies both conditions.

Question - 79
Swap and Unite

Implement a function that determines the minimum number of swaps required to make all occurrences of at least one character in a given string *s*, of length *n*, form a contiguous segment. That is, all occurrences of at least one letter are together.

In one operation, you can swap any two characters in the string.

The function *getMinSwaps* will take the following input:

string s: the given string.

The function should return an integer denoting the minimum number of swaps needed to achieve the desired condition.

Example

s = "abab"

Operations Table

Swap Indices (0-based)	Original String	New String	Character(s) forming Contiguous Segment
(1, 2)	"abab"	"aabb"	'a' and 'b'
(0, 2)	"abab"	"abab"	No character forms a contiguous segment
(0, 1)	"abab"	"baab"	'a'

Hence, the answer is 1.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- It is guaranteed that the string *s* consists of lowercase English letters only.

▼ Input Format for Custom Testing

The first line contains a string *s*, of size *n*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
aba     →     s = "aba"
```

Sample Output 0

```
0
```

Explanation

The character 'b' already forms a contiguous segment in the given string.

Hence, the answer is 0.

▼ Sample Case 1

Sample Input 1

```
STDIN             FUNCTION
-----             --------
ababab     →      s = "ababab"
```

Sample Output 1

```
1
```

Explanation

An optimal sequence of operations is:

- Swap the characters at indices (0-based indexing) (1, 4). The string becomes "aaabbb". The character 'a' is in a contiguous segment, and so is 'b'. Either one meets the criterion.

Hence, the answer is 1.

Question - 80
Maximize Array Sum

Implement a function that determines the maximum possible sum of an array *arr* of size *n* after performing the following operation any number of times (possibly zero).

An operation consists of:

1. Choosing two indices *l* and *r* such that $0 \le l < r < n$.
2. Changing *arr[i]* to min(*arr[l]*, *arr[r]*) for all $l < i < r$.

The function *getMaxSum* will take the following input:
  int arr[n]: the given array.

The function should return a long integer denoting the maximum possible sum of the array after performing the operations.

Example
$n = 5$
arr = [5, 1, 7, 1, 4]

An optimal sequence of operations (0-based indexing) is:

- Choosing $l = 0$ and $r = 2$. The new array is [5, 5, 7, 1, 4].
- Choosing $l = 2$ and $r = 4$. The new array is [5, 5, 7, 4, 4].

No further operations can increase the maximum sum of the array; hence, the answer is 25.

Constraints

- $1 \le n \le 2 * 10^5$
- $0 \le arr[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
4       →      arr[] size n = 4
5       →      arr = [5, 3, 1, 7]
3
1
7
```

Sample Output 0

```
22
```

Explanation
An optimal operation (0-based indexing) is:

- Choosing $l = 0$ and $r = 3$. The new array is [5, 5, 5, 7].

▼ **Sample Case 1**

Sample Input 1

```
STDIN         FUNCTION
-----         --------
2       →     arr[] size n = 2
2       →     arr = [2, 3]
3
```

Sample Output 1

```
5
```

Explanation

No operation can increase the sum of the array.

Question - 81
Priority Task Scheduling

In a task scheduling system, tasks are added dynamically and can be executed when their priority reaches a certain threshold.

Given a 2D array *operations* of size *m x 2*, implement a function to process all *m* operations. Each operation is one of the following types:

- [1, 0] – Add a new task with an initial priority of 0.
- [2, *t*] – Wait for *t* seconds. During this time, the priority of all scheduled tasks increases by *t*.
- [3, *p*] – Execute all tasks with a priority of at least *p* and return the number of tasks executed.

Note: The executed tasks are removed from the system and will not be considered for subsequent operations.

The function *executeTasks* should take the following inputs:
int operations[m][2]: A 2D array representing *m* operations.

The function should return an array containing the results of all type-3 operations, in the order they appear in the input. Assume that operations of type-1 and type-3 are completed instantaneously.

Example
operations = [[1, 0], [2, 4], [1, 0], [3, 3], [1, 0], [3, 7], [2, 5], [3, 5]]

There are a total of 3 tasks. The priority of i^{th} task is denoted by *priority[i]*. A priority value of -1 indicates that the task has already been executed.

operations[i]	time	*priority*	Explanation	Returns
[1, 0]	0	[0]	Task 1 is added for scheduling.	None
[2, 4]	4	[4]	The priority of Task 1 increases by 4.	None
[1, 0]	4	[4, 0]	Task 2 is added for scheduling.	None
[3, 3]	4	[-1, 0]	Task 1 is executed.	1
[1, 0]	4	[-1, 0, 0]	Task 3 is added for scheduling.	None
[3, 7]	4	[-1, 0, 0]	No tasks are executed.	0
[2, 5]	9	[-1, 5, 5]	The priority of Tasks 2 and 3 increases by 5.	None
[3, 5]	9	[-1, -1, -1]	Tasks 2 and 3 are executed.	2

The result is [1, 0, 2].

Constraints

- $2 \le m \le 2 * 10^5$
- $1 \le t \le 10^9$
- $0 \le p \le 10^9$
- It is guaranteed that there is at least one type-3 operation.

▼ Input Format for Custom Testing

The first line contains an integer m, the size of the array *operations[]*.
The second line contains the integer 2, the size of *operations[i]*.
Each of the next *m* lines contains two space separated integers, *operations[i][0]* and *operations[i][1]*, describing the i^{th} operation.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function
-----         --------
5      →      operations[] size m = 5
2      →      operations[][] size = 2
3 5    →      operations = [[3, 5], [1, 0], [1, 0], [2, 5], [3, 3]]
1 0
1 0
2 5
3 3
```

Sample Output 0

```
0
2
```

Explanation
There are 2 tasks. The priority of i^{th} task is denoted by *priority[i]*. A priority value of -1 indicates that the task has already been executed.

operations[i]	time	*priority*	Explanation	Returns
[3, 5]	0	[]	There are no tasks scheduled.	0
[1, 0]	0	[0]	Task 1 is added for scheduling.	None
[1, 0]	0	[0, 0]	Task 2 is added for scheduling	None
[2, 5]	5	[5, 5]	The priority of Tasks 1 and 2 increase by 5.	None
[3, 3]	5	[-1, -1]	Tasks 1 and 2 are executed.	2

The result is [0, 2].

▼ Sample Case 1

Sample Input 1

```
STDIN         Function
-----         --------
5      →      operations[] size m = 5
2      →      operations[][] size = 2
1 0    →      operations = [[1, 0], [2, 1], [1, 0], [3, 1], [3, 2]]
2 1
1 0
```

```
3 1
3 2
```

Sample Output 1

```
1
0
```

Explanation

There are 2 tasks. The priority of i^{th} task is denoted by *priority[i]*. A priority value of -1 indicates that the task has already been executed.

operations[i]	time	*priority*	Explanation	Returns
[1, 0]	0	[0]	Task 1 is added for scheduling.	None
[2, 1]	1	[1]	The priority of Task 1 increases by 1.	None
[1, 0]	1	[1, 0]	Task 2 is added for scheduling.	None
[3, 1]	1	[-1, 0]	Task 1 is executed.	1
[3, 2]	1	[-1, 0]	No task is executed.	0

The result is [1, 0].

Question - 82
Pair of Triplets

Implement a function that counts the number of unordered pairs of triplets in an array *arr* of size *n* that differ in exactly one position. A triplet is formed as [*arr[j]*, *arr[j+1]*, *arr[j+2]*] for each valid index *j* ($0 \le j \le n - 3$).

A pair of triplets *b* and *c* are said to differ in exactly one position if one of the following conditions holds (0-based indexing):

- $b[0] \ne c[0]$ and $b[1] = c[1]$ and $b[2] = c[2]$;
- $b[0] = c[0]$ and $b[1] \ne c[1]$ and $b[2] = c[2]$;
- $b[0] = c[0]$ and $b[1] = c[1]$ and $b[2] \ne c[2]$;

The function *countTripletPairs* will take one input:
int arr[n]: The elements of the array *arr*.

The function should return a long integer, denoting the count of unordered pairs of triplets that differ in exactly one position.

Example
arr = [3, 2, 2, 2, 3]

There are 3 possible triplets: [3, 2, 2], [2, 2, 2], [2, 2, 3].

Pair of triplets

First Triplet	Second Triplet
[3, 2, 2]	[2, 2, 2]
[2, 2, 2]	[2, 2, 3]

Therefore, 2 pairs of triplets differ in exactly one position. Hence, the answer = 2.

Constraints

- $3 \le n \le 2 * 10^5$

- $1 \le arr[i] \le 10^6$

▼ Input Format for Custom Testing

The first line contains the integer *n,* the size of array *arr*.

The next *n* lines contain an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN              Function
-----              --------
4          →       arr[] n = 4
2          →       arr = [2, 1, 1, 1]
1
1
1
```

Sample Output 0

```
1
```

Explanation

There are 2 possible triplets: [2, 1, 1], [1, 1, 1].

Pair of triplets

First Triplet	Second Triplet
[2, 1, 1]	[1, 1, 1]

Therefore, there is only 1 pair of triplets that differ in exactly one position. Hence, the answer = 1.

▼ Sample Case 1

Sample Input 1

```
STDIN                Function
-----                --------
8           →        arr[] n = 8
1           →        arr = [1, 2, 3, 2, 2, 3, 4, 2]
2
3
2
2
3
4
2
```

Sample Output 1

```
3
```

Explanation

There are 6 possible triplets: [1, 2, 3], [2, 3, 2], [3, 2, 2], [2, 2, 3], [2, 3, 4], [3, 4, 2].

Pair of triplets

First Triplet	Second Triplet
[1, 2, 3]	[2, 2, 3]
[2, 3, 2]	[2, 3, 4]
[3, 2, 2]	[3, 4, 2]

Therefore, 3 pairs of triplets differ in exactly one position. Hence, the answer = 3.

Question - 83
Monotonic Array Reduction

Implement a function that determines the number of operations required to reduce all elements of a given array *arr* to zero.

In each operation, traverse the whole array from left to right. For each index *i* (from 0 to *n-1*, using 0-based indexing):

- If *i* = *n-1* or *arr[i]* > *arr[i + 1]*, update *arr[i]* to max(0, *arr[i]* - 1).
- Otherwise, leave *arr[i]* unchanged.

The function *getNumberOfOperations* will take one input:
int arr[n]: the given array of positive integers.

The function should return an integer denoting the number of operations required to make all elements of *arr* equal to zero.

Example
n = 3
arr = [1, 3, 1]

Here, the underlined elements represent those changed during the operation and their modified values.

The sequence of operations

Operation Number	*arr* before	*arr* after
1	$[1, \underline{3}, \underline{1}]$	$[1, \underline{2}, \underline{0}]$
2	$[1, \underline{2}, 0]$	$[1, \underline{1}, 0]$
3	$[1, \underline{1}, 0]$	$[1, \underline{0}, 0]$
4	$[\underline{1}, 0, 0]$	$[\underline{0}, 0, 0]$

Hence, the answer is 4.

Constraints

- $1 \le n \le 2*10^5$
- $1 \le arr[i] \le 10^9$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN                          Function
-----                          --------
2                 →            arr[] size n = 2
2                 →            arr = [2, 1]
1
```

Sample Output 0

```
2
```

Explanation
Here, the underlined elements represent those changed during the operation and their modified values.

The sequence of operations

Operation Number	*arr* before	*arr* after
1	[2, 1]	[1, 0]
2	[1, 0]	[0, 0]

Hence, the answer is 2.

▾ Sample Case 1

Sample Input 1

```
STDIN                          Function
-----                          --------
3               →              arr[] size n = 3
3               →              arr = [3, 1, 2]
1
2
```

Sample Output 1

```
 4
```

Explanation
Here, the underlined elements represent those changed during the operation and their modified values.

The sequence of operations

Operation Number	*arr* before	*arr* after
1	[3, 1, 2]	[2, 1, 1]
2	[2, 1, 1]	[1, 1, 0]
3	[1, 1, 0]	[1, 0, 0]
4	[1, 0, 0]	[0, 0, 0]

Hence, the answer is 4.

Question - 84
Valid Binary Transformation

Implement a function that determines the minimum number of operations (possibly zero) required to transform a given binary string *s* of length *n* into a valid string. A string is considered valid if it does not contain "010" or "101" as a subsequence.

In each operation :

- Select an index in the string and flip the character at that index, i.e., change '0' to '1' or '1' to '0'.

The function *getMinimumOperations* will take one input :
    string s: the binary string

The function should return an integer denoting the minimum number of operations required to transform *s* into a valid string.

Note :

- A subsequence is a sequence formed by deleting zero or more characters from a string without changing the order of the remaining characters.

Example
s = "1010"

The following are some possible sequences of operations. Here, the underlined characters are those flipped during the operation.

- "1010" → "0010". This string is not valid since it contains "010" as a subsequence.
- "1010" → "1011" → "0011". This string is valid, and the number of operations required is 2.
- "1010" → "1000". This string is valid, and the number of operations required is 1.

Hence, the optimal answer is 1.

Constraints

- $1 \le n \le 2*10^5$
- The string *s* consists only of characters '0' and '1'

▼ Input Format for Custom Testing

The first and only line contains a binary string *s*.

▼ Sample Case 0

Sample Input 0

```
STDIN                          Function
-----                          --------
101              →             string s = "101"
```

Sample Output 0

```
1
```

Explanation
The following are some possible sequences of operations. Here, the underlined characters are those flipped during the operation.

- "101". This string is not valid since it contains "101" as a subsequence.
- "101" → "111". This string is valid, and the number of operations required is 1.

Hence, the optimal answer is 1.

▼ Sample Case 1

Sample Input 1

```
STDIN                          Function
-----                          --------
010110           →             string s = "010110"
```

Sample Output 1

```
2
```

Explanation
The following are some possible sequences of operations. Here, the underlined characters are those flipped during the operation.

- "010110" → "000110" → "001110" → "001111". This string is valid, and the number of operations required is 3.
- "010110" → "110110" → "110100". This string is not valid since it contains both "010" and "101" as a subsequence.
- "010110" → "010111" → "000111". This string is valid, and the number of operations required is 2.

Hence, the optimal answer is 2.

Question - 85
Equal String

You are given two strings *str1* and *str2*, each of length *n*, consisting of lowercase English letters.

You can repeat the following operation any number of times (possibly zero):

- Remove the first character of *str1* and insert it at any position of *str1*.

Implement a function that determines whether it is possible to make *str1* equal to *str2*, and if possible, find the minimum number of operations needed.

The function *getMinOperations* will take two inputs:
string str1: a string of length *n*.
string str2: a string of length *n*.

The function should return an integer denoting the minimum number of operations needed, or -1 in case it is impossible to make *str1* equal to *str2*.

Example
n = 4
str1 = "abab"
str2 = "abba"

In the following table, the underlined character is re-positioned during the operation.

Optimal Operations Table

Operation	Inserted at Index/Position (0-based index)	*str1* (After Operation)
1	3	baba
2	1	abba

A minimum of two operations is required, so the answer is 2.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- It is guaranteed that the strings *str1* and *str2* consist of lowercase English letters only.
- It is guaranteed that *length(str1) = length(str2).*

▼ Input Format for Custom Testing

The first line contains a string *str1*
The next line contains a string *str2*

▼ Sample Case 0

Sample Input 0

```
STDIN        FUNCTION
-----        --------
a       →    str1 = "a"
b       →    str2 = "b"
```

Sample Output 0

```
-1
```

Explanation
It is impossible to make *str1* equal to *str2*. Hence, return -1.

▼ Sample Case 1

Sample Input 1

```
STDIN        FUNCTION
-----        --------
abc     →    str1 = "abc"
cba     →    str2 = "cba"
```

Sample Output 1

```
2
```

Explanation

In the following table, the underlined character is re-positioned during the operation.

Optimal Operation Table

Operation	Inserted at Index/Position (0-based index)	*str1* (After Operation)
1	2	bc<u>a</u>
2	1	c<u>b</u>a

Question - 86
Element Grouping

Given an array of integers *nums* with *n* elements, we need to divide these elements into groups of exactly 3 elements each. Each element can belong to at most one group, and not all elements need to be used. For each valid group we form, we calculate its median value.

Implement a function that finds the sum of all these median values after optimally grouping the elements to maximize this sum.

The function *findMaxMedianSum* will take one input:
int nums[n]: the array

The function should return a long integer denoting the maximum total sum of medians of these groups.

Example
n = 5
nums = [2, 3, 4, 3, 4]

The division of array elements into groups

Groups	Values	Median
[2, <u>3</u>, <u>4</u>, 3, <u>4</u>]	[3, 4, 4]	4

The underlined elements form a group of 3 elements with a median of 4. The maximum sum of medians is 4.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le nums[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n,* the size of array *nums*.
The next *n* lines contain an integer *nums[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN               Function
-----               --------
6           →       nums[] size n = 6
3           →       nums = [3, 2, 9, 1, 6, 4]
2
9
1
6
4
```

Sample Output 0

```
9
```

Explanation

The division of array elements into groups

Groups	Values	Median
[3, 2, 9, 1, 6, 4]	[2, 6, 9]	6
[3, 2, 9, 1, 6, 4]	[1, 3, 4]	3

The underlined elements form groups. The maximum sum of medians is 6 + 3 = 9.

▼ Sample Case 1

Sample Input 1

```
STDIN               Function
-----               --------
7           →       nums[] size n = 7
8           →       nums = [8, 6, 3, 4, 4, 5, 6]
6
3
4
4
5
6
```

Sample Output 1

```
11
```

Explanation

The division of array elements into groups

Groups	Values	Median
[8, 6, 3, 4, 4, 5, 6]	[4, 5, 6]	5
[8, 6, 3, 4, 4, 5, 6]	[3, 6, 8]	6

The underlined elements form groups. The maximum sum of medians is 6 + 5 = 11.

Question - 87
Bit Flips

Implement a function that finds the lexicographically largest string that can be obtained after performing exactly *k* operations on a given binary string *s*.

- In one operation, a bit is selected, and all bits except that bit will get flipped (0 becomes 1, 1 becomes 0).

The function *findLargestString* takes two inputs:
string s: the source string
int k: the number of operations perform on *s*

The function should return the lexicographically largest string that can be obtained after performing *k* operations on *s*.

Note: A binary string *a* is lexicographically larger than a binary string *b* of the same length if and only if the following holds: in the first position where *a* and *b* differ, the string *a* contains '1', and the string *b* contains '0'.

Example
s = "101"
k = 2

Operations on the binary string

String before the operation	String after the operation
"1$\underline{0}$1"	"0$\underline{0}$0"
"00$\underline{0}$"	"11$\underline{0}$"

The underlined bit is chosen for the operation.

The lexicographically largest string possible is "110".

Constraints

- $1 \le |s| \le 2 * 10^5$
- It is guaranteed that *s* consists of characters '0' and/or '1'.
- $0 \le k \le 10^9$

▼ Input Format for Custom Testing

The first line contains the original string *s*.
The next line contains an integer *k*, the number of operations.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
1001      →    s = "1001"
5         →    k = 5
```

Sample Output 0

```
1110
```

Explanation

Operations on the binary string

String before operation	String after operation
"$\underline{1}$001"	"$\underline{1}$110"
"111$\underline{0}$"	"000$\underline{1}$"
"000$\underline{1}$"	"111$\underline{0}$"
"111$\underline{0}$"	"000$\underline{1}$"
"000$\underline{1}$"	"111$\underline{0}$"

The underlined bit is chosen for the operation.

▼ Sample Case 1

Sample Input 1

```
STDIN              Function
-----              --------
101101      →      s = "101101"
3           →      k = 3
```

Sample Output 1

```
111110
```

Explanation

Operations on the binary string

String before operation	String after operation
"101101"	"110010"
"110010"	"000101"
"000101"	"111110"

Question - 88
Computation Cost

There are *n* servers available for computation, each having its own computation capacity given in the array *computationCap*. The computation capacity of the *i*th server can be increased by one unit at an expense of *cost[i]*. The computation capacity of a server cannot exceed *maxCap*.

Implement a function to calculate the minimum cost required to adjust server computation capacities to achieve a minimum target average *avgCap*.

The function *minCostRequired* will take four inputs:
int computationCap[n]: current computation capacities of the servers
int cost[n]: the costs to increase the computation capacity of respective servers by one unit
int avgCap: the target average computation capacity
int maxCap: the server maximum computation capacity of the servers

The function should return the minimum cost required to adjust computation capacities to achieve the minimum target average. If this is not possible, return -1.

Note: The new average computation capacity must be greater than or equal to *avgCap*.

Example
computationCap = [2, 3, 5]
cost = [3, 2, 1]
avgCap = 6
maxCap = 10

The minimum total computation capacity needed is 6 * 3 = 18.
The current total computation capacity is the sum of all *compCap[]* i.e. 2 + 3 + 5 = 10.
The required increase in capacity is 18 - 10 = 8.
This can be done by increasing the capacity of servers 3 and 2 (1-based) by 5 and 3 units, respectively.
Therefore, minimum cost = 1*5 + 2*3 = 11.

Constraints

- $1 \le n \le 2 * 10^5$

- $1 \le avgCap, maxCap \le 10^6$
- $1 \le cost[i] \le 10^6$
- $1 \le computationCap[i] \le maxCap$

▼ Input Format for Custom Testing

The first line contains an integer *n,* the number of servers.
The next *n* lines contain an integer *computationCap[i]*.
The next line contains an integer *n,* the number of servers.
The next *n* lines contain an integer *cost[i]*.
The next line contains an integer *avgCap,* the needed average capacity of servers.
The next line contains an integer *maxCap,* the maximum capacity of a server.

▼ Sample Case 0

Sample Input 0

```
STDIN               Function
-----               --------
3           →       computationCap[] size n = 3
1           →       computationCap = [1, 1, 1]
1
1
3           →       cost[] size n = 3
10          →       cost = [10, 7, 4]
7
4
2           →       avgCap = 2
5           →       maxCap = 5
```

Sample Output 0

```
12
```

Explanation

The minimum total computation capacity needed is 2 * 3 = 6.
The current total computation capacity is the sum of all *computationCap[]* i.e. 1 + 1 + 1 = 3.
The required increase in capacity is 6 - 3 = 3.
This can be done by increasing the capacity of server 3 (1-based) by 3 units.
Therefore, minimum cost = 3*4 = 12.

▼ Sample Case 1

Sample Input 1

```
STDIN               Function
-----               --------
5           →       computationCap[] size n = 5
5           →       computationCap = [5, 4, 3, 3, 2]
4
3
3
2
5           →       cost[] size n = 5
2           →       cost = [2, 7, 1, 2, 5]
7
1
2
5
4           →       avgCap = 4
5           →       maxCap = 5
```

Sample Output 1

```
4
```

Explanation

The minimum total computation capacity needed is 5 * 4 = 20.
The current total computation capacity is the sum of all *computationCap[]* i.e. 5 + 4 + 3 + 3 + 2 = 17.
The required increase in capacity is 20 - 17 = 3.
This can be done by increasing the capacity of servers 3 and 4 (1-based) by 2 and 1 units, respectively.
Therefore, minimum cost = 1*2 + 2*1 = 4.

Question - 89
Streaming Groups

A streaming platform allows *n* users to create watch parties where multiple users can watch content together. Each user has an engagement score represented by the array *engagementScores*.

A valid watch party must satisfy the following conditions:

- The product of the number of participants in the party and the minimum engagement score within that party must be at least *minWatchPartyScore*.
- Each user can be in at most one watch party, and some users may remain ungrouped.

Implement a function that determines the maximum number of watch parties that can be formed.

The function *findMaxWatchParties* will take two inputs:
int engagementScores[n]: the engagement scores of the users
int minWatchPartyScore: the minimum required product for a valid watch party

The function should return an integer, representing the maximum number of watch parties that can be formed.

Example
n = 3
engagementScores = [3, 4, 5]
minWatchPartyScore = 5

Calculate all possible watch party

User Indices (0-based)	Calculation	Valid
{0, 1, 2}	3 × 3 = 9	Yes
{0, 1}	2 × 3 = 6	Yes
{0, 2}	2 × 3 = 6	Yes
{0}	1 × 3 = 3	No
{1}	1 × 4 = 4	No
{1, 2}	2 × 4 = 8	Yes
{2}	1 × 5 = 5	Yes

Since each user can be in at most one watch party, the goal is to maximize the number of valid watch parties.
One optimal way is to form two watch parties:

- One watch party consists of user 0 and user 1.
- One watch party consists of user 2.

Hence, the answer is 2.

Constraints

- $1 \le n \le 2 * 10^5$

- $1 \le$ engagementScores*[i]* $\le 10^9$
- $1 \le$ *minWatchPartyScore* $\le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the number of users.
Each of the next *n* lines contains an integer *engagementScores[i]*.
The next line contains an integer *minWatchPartyScore*.

▼ Sample Case 0

Sample Input 0

```
STDIN       FUNCTION
-----       --------
4       →   engagementScores[] size n = 4
2       →   engagementScores = [2, 4, 2, 3]
4
2
3
1       →   minWatchPartyScore = 1
```

Sample Output 0

```
4
```

Explanation
Each user can join their own watch party since their engagement score is greater than *minWatchPartyScore*. Hence, the answer is 4.

▼ Sample Case 1

Sample Input 1

```
STDIN        FUNCTION
-----       --------
2       →   engagementScores[] size n = 2
2       →   engagementScores = [2, 1]
1
2       →   minWatchPartyScore = 2
```

Sample Output 1

```
1
```

Explanation
Only one valid watch party can be formed. Either:

- Include both users in one party, or
- Only include the user with an engagement score of 2 in a single party.

Hence, the answer is 1.

Question - 90
Subarray Sum

Implement a function that finds the maximum subarray sum of the array *arr1* of length *n* after performing the following operation exactly *m* times.
Given an array *arr2* of size *m*:

- Choose either the first or the last element of array *arr2*.
- Insert the chosen element to either the front or the back of array *arr1*.
- Delete the chosen element from array *arr2*.

The function *getMaxSum* will take two inputs:

int arr1[n]: the array *arr1*
int arr2[m]: the array *arr2*

The function should return an integer representing the maximum sum of any subarray of the array *arr1* that you can achieve after performing exactly *m* operations.

Note: An array *b* is a subarray of the array *a* if *b* can be obtained from *a* by deleting several (possibly zero or all) elements from the beginning and several (possibly zero or all) elements from the end.

Example

n = 5
arr1 = [1, 3, -5, 2, 2]
m = 2
arr2 = [2, 3]

Operations	Updated *arr1*	Updated *arr2*
Add the first element of array *arr2* to the back of array *arr1*.	[1, 3, -5, 2, 2, 2]	[3]
Add the first element of array *arr2* to the back of array *arr1*.	[1, 3, -5, 2, 2, 2, 3]	[]

The maximum sum subarray of array *arr1* is [2, 2, 2, 3], with a sum of 2 + 2 + 2 + 3 = 9.
Hence, the answer is 9.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ *m* ≤ 2 * 10^5
- -10^8 ≤ *arr1[i]*, *arr2[i]* ≤ 10^8

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr1*.
Each of the next *n* lines contains an integer *arr1[i]*.
The next line contains an integer *m*, the size of the array *arr2*.
Each of the next *m* lines contains an integer *arr2[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
1         →    arr1[] size n = 1
10        →    arr1 = [10]
1         →    arr2[] size m = 1
-10       →    arr2 = [-10]
```

Sample Output 0

```
10
```

Explanation

Operations	Updated *arr1*	Updated *arr2*
Add the first element of array *arr2* to the front of array *arr1*.	[-10, 10]	[]

The maximum sum subarray of array *arr1* is [10] having a sum of 10.

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
5          →    arr1[] size n = 5
10         →    arr1 = [10, -5, 14, -20, 4]
-5
14
20
4
3          →    arr2[] size m = 3
-10        →    arr2 = [-10, 5, -2]
5
-2
```

Sample Output 1

```
24
```

Explanation

Operations	Updated *arr1*	Updated *arr2*
Add the last element of array *arr* to the back of array *arr1.*	[10, -5, 14, -20, 4, -2]	[-10, 5]
Add the last element of array *arr2* to the front of array *arr1*.	[5, 10, -5, 14, -20, 4, -2]	[-10]
Add the first element of array *arr2* to the front of array *arr1*.	[-10, 5, 10, -5, 14, -20, 4]	[]

The maximum sum subarray of array *arr1* is [5, 10, -5, 14], with a sum of 5 + 10 - 5 + 14 = 24.

Question - 91
Count Subarrays

Implement a function that counts subarrays within an array *arr* of length *n* where the subarray's length appears as one of its elements.

The function *getSubarrays* will take one input:
int arr[n]: elements of the arr

The function should return an integer representing the number of such subarrays.

Note: An array *b* is a subarray of the array *a* if *b* can be obtained from *a* by deleting several (possibly zero or all) elements from the beginning and/or the end.

Example
n = 3
arr = [1, 2, 1]

Checking Valid Subarrays

Subarray	Length	Condition Met?
[1]	1	Yes
[2]	1	No

[**1**]	1	Yes
[1, **2**]	2	Yes
[**2**, 1]	2	Yes
[1, 2, 1]	3	No

Hence, the number of such subarrays is 4.

Constraints

- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ *arr[i]* ≤ *n*

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr[]*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
5         →    arr[] size n = 5
2         →    arr = [2, 3, 1, 3, 5]
3
1
3
5
```

Sample Output 0

```
6
```

Explanation

Checking Valid Subarrays

Subarray	Length
[**1**]	1
[**2**, 3]	2
[2, **3**, 1]	3
[**3**, 1, 3]	3
[1, **3**, 5]	3
[2, 3, 1, 3, **5**]	5

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
6         →    arr[] size n = 6
1         →    arr = [1, 1, 2, 2, 3, 3]
1
2
```

```
2
3
3
```

Sample Output 1

```
7
```

Explanation

Checking Valid Subarrays

Subarray	Length
[**1**]	1
[**1**]	1
[1, **2**]	2
[**2**, 2]	2
[**2**, 3]	2
[2, 2, **3**]	3
[2, **3**, 3]	3

Question - 92
Server Memory Allocation

Given *n* programs and *n–1* servers, each program must run on its own server, and each server can only handle one program. Every program has a memory requirement given by the array *memoryRequirement*. Every server has a fixed memory capacity, given by the array *memoryAvailable*. Since there is one extra program, a new server must be added.

Implement a function to determine the smallest memory capacity required for the new server to ensure that all programs can be executed.

The function *getMinMemory* takes the following inputs:
  int memoryRequirement[n]: the memory requirements of the programs
  int memoryAvailable[n-1]: the memory capacities of the existing servers

The function should return the minimum memory size required for the new server or -1 if allocation is impossible.

Example:
memoryRequirement = [5, 2, 3, 7]
memoryAvailable = [6, 2, 8].

If the new server's memory capacity is less than 3, not all programs can be assigned to servers. Add a new server with a capacity of 3 to allocate the programs as follows:

Program	Program Requirement	Assigned Server (1-based)	Server Capacity
Program 1	5	Server 1	6
Program 2	2	Server 2	2

Program 3	3	New Server	3
Program 4	7	Server 3	8

Hence, the answer is 3

Constraints:

- 1 ≤ *n* ≤ 2 * 10^5
- 1 ≤ *memoryRequirement[i]* , *memoryAvailable[j]* ≤ 10^9

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *memoryRequirement*.
Each of the next *n* lines contains an integer *memoryRequirement[i]*.
The next line contains an integer *n-1*, the size of the array *memoryAvailable*.
Each of the next *n-1* lines contains an integer *memoryAvailable[j]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function
-----         --------
4       →     memoryRequirement[]  size n = 4
3       →     memoryRequirement = [3, 7, 2, 5]
7
2
5
3       →     memoryAvailable[] size = 3
8       →     memoryAvailable = [8, 1, 6]
1
6
```

Sample Output 0

```
-1
```

Explanation
No matter what the memory capacity of the new server is, it is impossible to assign all the programs to the servers.

▼ Sample Case 1

Sample Input 1

```
STDIN         Function
-----         --------
5       →     memoryRequirement[] size n = 5
2       →     memoryRequirement = [2, 2, 7, 3, 5]
2
7
3
5
4       →     memoryAvailable[] size = 4
3       →     memoryAvailable = [3, 2, 7, 2]
2
7
2
```

Sample Output 1

```
5
```

Explanation
If the new server's memory capacity is less than 5, not all programs can be assigned to servers. Add a new server with a capacity of 5 to allocate the programs as follows:

Program	Program Requirement	Assigned Server (1-based)	Server Capacity
Program 1	2	Server 4	2
Program 2	2	Server 2	2
Program 3	7	Server 3	7
Program 4	3	Server 1	3
Program 5	5	New Server	5

Question - 93
Optimize the String

Implement a function that finds the lexicographically smallest string achievable after performing the following operations any number of times on the string *s* of length *n*:

- Replace any substring of length 3 with "100" without changing the order of other characters.

The function *getMinString* will take one input:
string s: a string *s*.

The function should return the lexicographically smallest string possible.

Note: A string *a* is a substring of a string *b* if *a* can be obtained from *b* by the deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Example
s = "01010"

Different ways to apply operations:

Operations on the string

Transformation	Explanation
01010 → 01000	Replace "101" (at index 1-3 (0-based indexing)) with "100".
01010 → 01100	Replace "010" (at index 2-4 (0-based indexing)) with "100".
01010 → 10010	Replace "010" (at index 0-2 (0-based indexing)) with "100".

Thus, the lexicographically smallest string achievable is "01000".

Constraints

- $3 \leq n \leq 2 * 10^5$
- It is guaranteed that string *s* consists of characters '0' and '1'.

▼ Input Format for Custom Testing

The first line contains a string *s.*

▼ **Sample Case 0**

Sample Input 0

```
STDIN          FUNCTION
-----          --------
0111      →     s = "0111"
```

Sample Output 0

```
0100
```

Explanation

- Replace "111" with "100": After replacement, the string becomes: *s* = "0100".
- No further operations produce a lexicographically smaller string.

▼ **Sample Case 1**

Sample Input 1

```
STDIN          FUNCTION
-----          --------
0001      →     s = "0001"
```

Sample Output 1

```
0001
```

Explanation

Since no replacement reduces the lexicographical order of the string, no operations are performed.

Question - 94
First Compatible

The task is to implement a function that, for each element in the first array (*arr1*), finds the 1-based index of the first element in the second array (*arr2*) that is greater than it.

The function *getFirstCompatible* will take the following two inputs:
  int arr1[n]: the array *arr1*.
  int arr2[m]: the array *arr2*.

The function should return an array of length *n*, where the i^{th} entry corresponds to the smallest index *j* (1-based indexing) in *arr2* such that *arr1[i]* < *arr2[j]*. If no such index *j* exists, return -1.

Example
n = 3
arr1 = [3, 8, 1]
m = 3
arr2 = [1, 2, 5].

Explanation

i	*arr1[i]*	Comparison Details (1-based indexing)	Result
1	3	*arr2[3]* = 5 is the smallest index having value > 3.	3
2	8	No index in *arr2* has value greater than 8.	-1
3	1	*arr2[2]* = 2 is the smallest index having value > 1.	2

Hence, the output is [3, -1, 2].

Constraints

- $1 \le n, m \le 2 * 10^5$
- $-10^9 \le arr1[i], arr2[j] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr1*.
Each of the next *n* lines contains an integer *arr1[i]*.
The next line contains an integer *m*, the size of the array *arr2*.
Each of the next *m* lines contains an integer *arr2[j]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function
-----         --------
4       →     arr1[] size n = 4
5       →     arr1 = [5, 1, 7, 4]
1
7
4
4       →     arr2[] size m = 4
1       →     arr2 = [1, 2, 3, 5]
2
3
5
```

Sample Output 0

```
-1
2
-1
4
```

Explanation

Explanation

i	*arr1[i]*	Comparison Details	Result
1	5	No index in *arr2* has a value > 5.	-1
2	1	*arr2[2]* = 2 is the smallest index with a value > 1.	2
3	7	No index in *arr2* has a value > 7.	-1
4	4	*arr2[4]* = 5 is the smallest index with a value > 4.	4

▼ Sample Case 1

Sample Input 1

```
STDIN         Function
-----         --------
5       →     arr1[] size n = 5
2       →     arr1 = [2, 8, 7, 4, 6]
8
7
4
6
4       →     arr2[] size m = 4
3       →     arr2 = [3, 5, 7, 3]
```

```
5
7
3
```

Sample Output 1

```
1
-1
-1
2
3
```

Explanation

Explanation

i	*arr1[i]*	Comparison Details	Result
1	2	*arr2[1]* = 3 is the smallest index with a value > 2.	1
2	8	No index in *arr2* has a value > 8.	-1
3	7	No index in *arr2* has a value > 7.	-1
4	4	*arr2[2]* = 5 is the smallest index with a value > 4.	2
5	6	*arr2[3]* = 7 is the smallest index with a value > 6.	3

Question - 95
Optimal Division

Implement a function that finds the minimum number of subsequences that you must divide the array *arr* of length *n* into such that:

- Each element is included in exactly one subsequence.
- Every subsequence must contain a majority element.

An integer *x* is the majority of sequence *b[]* if it occurs strictly more than |*b*| / 2 times, where |*b*| represents the length of *b*.

The function *getMinDivision* takes one input:
int arr[n]: the array

The function should return an integer representing the minimum number of divisions required.

Note: A subsequence of a string is a string that can be obtained by removing several (possibly zero) characters from the original string.

Example
n = 4
arr = [1, 2, 3, 4]

Since all elements are different, they must be in their own groups to represent a majority. The optimal and only valid division is [1], [2], [3], and [4]. Hence, the answer is 4.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le arr[i] \le n$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr[]*.

Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
4         →     arr[] size n = 4
1         →     arr = [1, 1, 1, 2]
1
1
2
```

Sample Output 0

```
1
```

Explanation

The whole array can be taken as one sequence with 1 as its majority element.

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
6         →     arr[] size n = 4
1         →     arr = [1, 1, 2, 2, 3, 3]
1
2
2
3
3
```

Sample Output 1

```
2
```

Explanation

Grouping elements into two subsequences, ensuring the condition of a majority element:

- Subsequence 1: [1, 1, 2] → Majority element: 1 (appears 2 times, strictly more than 3/2 = 1.5).
- Subsequence 2: [2, 3, 3] → Majority element: 3 (appears 2 times, strictly more than 3/2 = 1.5).

Question - 96
Grid Route Optimization

In a 2-D grid, you start at the coordinate *(startX, startY)* and want to reach the target coordinate *(endX, endY)*. Yu are given a string called *directions* of length *n*, which can contain the following characters:

- 'E' (East): Move to *(x + 1, y)* (right).
- 'S' (South): Move to *(x, y - 1)* (down).
- 'W' (West): Move to *(x - 1, y)* (left).
- 'N' (North): Move to *(x, y + 1)* (up).

Implement a function to calculate the earliest time required to reach from *(startX, startY)* to *(endX, endY)*.

At the i^{th} second, you can perform either of the two moves:

- Move one unit in the direction as specified by the character *directions[i]*.
- Stay where you are.

The function *getMinimumTime* will take the following parameters:

string directions: A string consisting of the directions ('E,' 'S,' 'W,' 'N') indicating the allowed direction for each second.
int startX: the starting X coordinate
int startY: the starting Y coordinate
int endX: the target X coordinate
int endY: the target Y coordinate

The function should return the earliest time (an integer) at which you can reach the target position *(endX, endY)*. If you cannot reach the target position within the given time *n*, return -1.

Example
directions = "WWNNSSE"
startX = 1
startY = -1
endX = -1
endY = -3

An optimal sequence of events

Time	Action	New Coordinates
t = 1	Move West	(0, -1)
t = 2	Move West	(-1, -1)
t = 3	Stay at the same place	(-1, -1)
t = 4	Stay at the same place	(-1, -1)
t = 5	Move South	(-1, -2)
t = 6	Move South	(-1, -3)

Hence, after 6 seconds, you reach (-1, -3) from (1, -1).

Constraints

- $1 \le n \le 2 * 10^5$
- $-10^9 \le startX, startY, endX, endY \le 10^9$

▾ Input Format for Custom Testing

The first line contains a string, *directions*.
The second line contains an integer, the starting X coordinate, *startX*.
The third line contains an integer, the starting Y coordinate, *startY*.
The fourth line contains an integer, the target X coordinate, *endX*.
The last line contains an integer, the target Y coordinate, *endY*.

▾ Sample Case 0

Sample Input 0

```
STDIN               Function
-----               --------
EWS         →       directions = "EWS"
1           →       startX = 1
2           →       startY = 2
2           →       endX = 2
1           →       endY = 1
```

Sample Output 0

```
3
```

Explanation

An optimal sequence of events

Time	Action	New Coordinates
t = 1	Move East	(2, 2)
t = 2	Stay at the same place	(2, 2)
t = 3	Move South	(2, 1)

Hence, after 3 seconds, you can reach (2, 1) from (1, 2).

▾ Sample Case 1

Sample Input 1

```
STDIN                  Function
-----                  --------
E          →           directions = "E"
1          →           startX = 1
1          →           startY = 1
2          →           endX = 2
2          →           endY = 2
```

Sample Output 1

```
-1
```

Explanation

If you go east, the coordinates are (2, 1); if you do not, the coordinates remain (1, 1). In either case, you cannot reach (2, 2). Hence, the answer is -1.

Question - 97
Optimal Operations

Implement a function that returns the minimum number of operations needed to ensure the string *s* (of length *n*) contains no segments of exactly *m* consecutive '0's.

In one operation, you can do the following:

- Select a contiguous segment of length *k* and make every bit in this segment '1'.

The function *getMinOperations* will take three inputs:
  string s: a string representing *s*.
  int m: an integer representing *m*.
  int k: an integer representing *k*.

The function should return an integer representing the minimum number of operations needed.

Example
s = "000000"
m = 3
k = 2

We can perform an operation on the interval [3, 4] (1-based indexing) to get "001100", ensuring no segment of consecutive 0s has a length ≥ 3. Thus, the answer is 1.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le m, k \le n$
- It is guaranteed that each character in *s* is '0' or '1'.

▼ Input Format for Custom Testing

The first line contains a string *s*.
The next line contains an integer *m*.
The next line contains an integer *k*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
10101     →     s = "10101"
1         →     m = 1
1         →     k = 1
```

Sample Output 0

```
2
```

Explanation

In one operation, we can flip only one bit (*k* = 1), so:

- Flip the 0 at position 2 (1-based indexing): 10101 → 11101
- Flip the 0 at position 4 (1-based indexing): 11101 → 11111

Hence, the number of operations required will be 2.

▼ Sample Case 1

Sample Input 1

```
STDIN          FUNCTION
-----          --------
10101     →     s = "10101"
2         →     m = 2
3         →     k = 3
```

Sample Output 1

```
0
```

Explanation

No operations are needed as the string already satisfies the condition.

Question - 98
Minimize the Operations

Implement a function that finds the minimum number of operations needed to empty the array *arr* of length *n*.

In one operation, you can do the following:

- Select an inclusive range of indices [*l*, *r*] (*l* ≤ *r*).
- Remove all elements equal to *min*(*arr[l], arr[l+1], …, arr[r]*) within that range.

Before doing the operations, you can perform up to *k* modifications on the array by changing the value at any index *i* (0 ≤ *i* < *n*) to any integer *p*.

The function *getMinOperations* will take two inputs:
  int arr[n]: an array *arr*
  int k: the number of modifications that can be performed on the array

The function should return an integer representing the minimum number of operations to empty the array.

Example
n = 3
arr = [2, 3, 2]
k = 1

Modification of the array

Array Before	Modification	Array After
[2, 3, 2]	Choose index *i* = 1 (0-based indexing) and modify *arr*[1] = 2.	[2, 2, 2]

Operations on the array

Array Before	Operation	Array After
[2, 2, 2]	Choose [0, 2] (0-based indexing). The minimum value is 2. Remove all 2s.	[]

Hence, the minimum number of operations after modifications up to *k* = 1 is 1.

Constraints
- $1 \le n \le 2 * 10^5$
- $0 \le k \le n$
- $1 \le arr[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr[]*.
Each of the next *n* lines contains an integer *arr[i]*.
The next line contains an integer *k*.

▼ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
4          →    arr[] size n = 4
1          →    arr = [1, 2, 3, 4]
2
3
4
2          →    k = 2
```

Sample Output 0

```
2
```

Explanation

Modification of the array

Array Before	Modification	Array After
[1, 2, 3, 4]	Choose index *i* = 1 (0-based indexing) and modify *arr*[1] = 1.	[1, 1, 3, 4]
[1, 1, 3, 4]	Choose index *i* = 2 (0-based indexing) and modify *arr*[2] = 1.	[1, 1, 1, 4]

Operations on the array

Array Before	Operation	Array After
[1, 1, 1, 4]	Choose [0, 3] (0-based indexing). The minimum value is 1. Remove all 1s.	[4]
[4]	Choose [0, 0] (0-based indexing). The minimum value is 4. Remove all 4s.	[]

Hence, the minimum number of operations after modifications up to *k* = 2 is 2.

▼ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
5        →     arr[] size n = 5
3        →     arr = [3, 1, 2, 1, 3]
1
2
1
3
0        →      k = 0
```

Sample Output 1

```
3
```

Explanation

Operations on the array

Array Before	Operation	Array After
[3, 1, 2, 1, 3]	Choose [1, 3] (0-based indexing). The minimum value is 1. Remove all 1s.	[3, 2, 3]
[3, 2, 3]	Choose [1, 1] (0-based indexing). The minimum value is 2. Remove all 2s.	[3, 3]
[3, 3]	Choose [0, 1] (0-based indexing). The minimum value is 3. Remove all 3s.	[]

Hence, the minimum number of operations after no modifications is 3.

Question - 99
Maximum Sum

You are given an array *arr* of size *n* and an integer *k* representing the number of sign-flip operations to perform. A sign-flip operation changes the sign of a single element in the array (from positive to negative or vice versa).

Implement a function that calculates the maximum possible sum of the array after performing exactly *k* sign-flip operations.

The function *maximizeSum* will take two inputs:
int arr[n]: the array
int k: the total number of sign-flip operations to perform

The function should return the maximum possible sum of the array after performing exactly *k* flips. A number's sign can be flipped multiple times as long as the total number of flips is exactly *k*.

Example

n = 5

arr = [-5, -2, -3, 6, 7]

k = 3

Some possible ways to select elements

arr before	*arr* after	Total sum
[-5, -2, -3, 6, 7]	[5, 2, 3, 6, 7]	5 + 2 + 3 + 6 + 7 = 23
[-5, -2, -3, 6, 7]	[-5, 2, 3, -6, 7]	-5 + 2 + 3 + (-6) + 7 = 1
[-5, -2, -3, 6, 7]	[5, 2, -3, -6, 7]	5 + 2 + (-3) + (-6) + 7 = 5

The underlined elements are the ones going through a sign flip. The maximum possible sum is 23.

Constraints

- $1 \le n \le 2 * 10^5$
- $-10^9 \le arr[i] \le 10^9$
- $1 \le k \le 10^9$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the length of the given array *arr*.

Each of the next *n* lines contains an integer *arr[i]*.

The last line contains the integer *k*.

▾ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
4         →    arr[] size n = 4
4         →    arr = [4, 2, 1, 9]
2
1
9
1         →    k = 1
```

Sample Output 0

```
14
```

Explanation

Some possible ways to select elements

arr before	*arr* after	Total sum
[4, 2, 1, 9]	[-4, 2, 1, 9]	-4 + 2 + 1 + 9 = 8
[4, 2, 1, 9]	[4, -2, 1, 9]	4 + (-2) + 1 + 9 = 12
[4, 2, 1, 9]	[4, 2, -1, 9]	4 + 2 + (-1) + 9 = 14

[4, 2, 1, 9]	[4, 2, 1, -9]	4 + 2 + 1 + (-9) = -2

The underlined elements are the ones that flip.

▾ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
5        →     arr[] size n = 5
3        →     arr = [3, -2, -1, -3, 10]
-2
-1
-3
10
2        →     k = 2
```

Sample Output 1

```
17
```

Explanation

Some possible ways to select elements

arr before	*arr* after	Total sum
[3, -1, -2, -3, 10]	[3, 1, -2, 3, 10]	3 + 1 + (-2) + 3 + 10 = 15
[3, -1, -2, -3, 10]	[3, 1, 2, -3, 10]	3 + 1 + 2 + (-3) + 10 = 13
[3, -1, -2, -3, 10]	[3, -1, 2, 3, 10]	3 + (-1) + 2 + 3 + 10 = 17

Question - 100
Cyclic Shift Validation

Given a string *s* of length *n,* it is considered balanced if its left cyclic shift is equal to its right cyclic shift.

Cyclic Shift Definitions:

- Left cyclic shift: Remove the first character and append it to the end.
- Right cyclic shift: Remove the last character and insert it at the beginning.

Implement a function that calculates the minimum number of characters that need to be removed (possibly zero) from *s* so that the resulting string is balanced.

The function *getMinimumRemovals* will take one input :
string s: the given string.

The function should return an integer, the minimum number of characters (possibly zero) to remove to balance the string.

Example
s = "74819"

Some possible ways of removing characters

s before	*s* after	Left cyclic shift	Right cyclic shift	Is the string balanced?
"74819"	"789"	"897"	"978"	No
"74819"	"49"	"94"	"94"	Yes
"74819"	"7481"	"4817"	"1748"	No
"74819"	"7"	"7"	"7"	Yes

Here, the underlined characters represent those removed during the operation.
A minimum of 3 characters, 7, 8, and 1, must be removed to make *s* balanced.
Hence, the answer is 3.

Constraints

- $2 \le n \le 2 * 10^5$
- It is guaranteed that the string *s* consists of digits from 0 to 9.

▾ Input Format for Custom Testing

The first line contains a string *s*, consisting of digits from 0 to 9.

▾ Sample Case 0

Sample Input 0

```
STDIN           FUNCTION
-----           --------
133       →     s = "133"
```

Sample Output 0

```
1
```

Explanation

Some possible ways of removing characters

s before	*s* after	Left cyclic shift	Right cyclic shift	Is the string balanced?
"133"	"13"	"31"	"31"	Yes
"133"	"33"	"33"	"33"	Yes
"133"	"133"	"331"	"313"	No

At least 1 character must be removed.

▾ Sample Case 1

Sample Input 1

```
STDIN           FUNCTION
-----           --------
474747    →     s = "474747"
```

Sample Output 1

```
0
```

Explanation

For this string, left cyclic shift = "747474" and right cyclic shift = "747474", so no characters need to be removed.

Question - 101
Minimal Subsegment Deletion

Given an array *arr* of size *n*. The goal is to remove at most one subsegment (a consecutive sequence of elements) so that all remaining elements are pairwise distinct, i.e., there are no duplicates.

Implement a function that determines the minimum size of at most one contiguous subsegment that can be removed in order to meet the required condition.

The function *getMinimumLength* will take one input:
int arr[n]: the elements of the array

The function should return an integer indicating the minimum size of continuous subsegment (possibly zero) that can be removed to meet the required condition.

Note: The array can become empty after the deletion.

Example
n = 4
arr = [3, 3, 3, 5]

Here, the subsegment to be removed is given in 1-based indexing and the underlined elements are the ones present in this subsegment.

Some possible ways of removing subsegments

Subsegment to be deleted	*arr[]* before	*arr[]* after	Is the condition met?
[1, 2]	[3, 3, 3, 5]	[3, 5]	Yes
[3, 4]	[3, 3, 3, 5]	[3, 3]	No, since 3 occurs twice.
[4, 4]	[3, 3, 3, 5]	[3, 3, 3]	No, since 3 occurs thrice.

It can be seen that removing a subsegment of length less than 2 cannot result in meeting the required condition.
Hence, the answer is 2.

Constraints
- $1 \le n \le 2000$
- $1 \le arr[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the length of the given array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN           Function
-----           --------
3         →     arr[] size n = 3
2         →     arr = [2, 1, 2]
1
2
```

Sample Output 0

```
1
```

Explanation
Here, the subsegment to be removed is given in 1-based indexing and the underlined elements are in the subsegment.

Some possible ways of removing subsegments

Subsegment to be deleted	*arr[]* before	*arr[]* after	Is the condition met?
[1, 1]	[2, 1, 2]	[1, 2]	Yes
[1, 2]	[2, 1, 2]	[2]	Yes
[2, 2]	[2, 1, 2]	[2, 2]	No, since 2 occurs twice.

A subsegment of at least length 1 must be removed.

▾ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
5        →     arr[] size n = 5
2        →     arr = [2, 3, 2, 3, 5]
3
2
3
5
```

Sample Output 1

```
2
```

Explanation
Here, the subsegment to be removed is given in 1-based indexing and the underlined elements are the ones present in this subsegment.

Some possible ways of removing subsegments

Subsegment to be deleted	*arr[]* before	*arr[]* after	Is the condition met?
[1, 1]	[2, 3, 2, 3, 5]	[3, 2, 3, 5]	No, since 3 occurs twice.
[1, 2]	[2, 3, 2, 3, 5]	[2, 3, 5]	Yes
[2, 4]	[2, 3, 2, 3, 5]	[2, 5]	Yes
[4, 5]	[2, 3, 2, 3, 5]	[2, 3, 2]	No, since 2 occurs twice.
[5, 5]	[2, 3, 2, 3, 5]	[2, 3, 2, 3]	No, since both 2 and 3 occur twice.

A subsegment of at least length 2 must be removed.

Question - 102
Minimize Transaction Ledger

A banking system processes transactions to maintain an efficient ledger. Each transaction is denoted by a lowercase English letter in the string *transactionLedger*.

- In one processing step, the system chooses the transaction at the i^{th} position and removes the first occurrence of the same transaction type to its left and to its right, if they exist.

Implement a function that determines the minimum number of processing steps required to minimize the ledger's length.

The function *calculateMinProcessingSteps* will take one input:
string transactionLedger: the transactions ledger

The function should return the minimum number of processing steps required to minimize the ledger's length.

Example
transactionLedger = "baabacaa".

The following moves are optimal.

1. Choose index 0, *"baa**b**acaa",* then *transactionLedger = "baaacaa".* Delete the *b* to its right at index 3. There is no *b* to its left, so the operation is finished.
2. Now, choose 'a' at index 2, remove the values at indices 1 and 3, *"b**a**a**a**caa",* and *transactionLedger = "bacaa".*
3. Now. choose index 3, remove the values at indices 1 and 4, *"b**a**ca**a**",* and *transactionLedger = "bca".*

The *transactionLedger* cannot be reduced further. Hence, return 3.

Constraints

- 1 ≤ |*transactionLedger*| ≤ 10^5
- The string *transactionLedger* consists of lowercase English letters.

▼ Input Format For Custom Testing

The first line of input contains a string that represents *transactionLedger*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN              FUNCTION
-----              --------
cbaa         →    transactionLedger = "cbaa"
```

Sample Output

```
1
```

Explanation
Optimally, choose 'a' at index 2 or 3, then *transactionLedger = "cba".*

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN              FUNCTION
-----              --------
abaaa         →    transactionLedger = "abaaa"
```

Sample Output

```
2
```

Explanation
An optimal sequence:

1. Choose 'a' at index 2, then *transactionLedger = "baa".*
2. Choose 'a' at index 1, then *transactionLedger = "ba".*

The *transactionLedger* cannot be reduced further.

Question - 103
Substring Transformation Cost

In a URL shortening service, two strings that each contain *n* lowercase English letters are provided: *originalUrl* and *targetUrl*. Additionally, an integer *cost* is given, representing the maximum allowed cost to transform a substring of *originalUrl* into the corresponding substring of *targetUrl*.

The cost of changing a character from *originalUrl[i]* to *targetUrl[i]* is the absolute difference of their ASCII values, i.e., abs(*originalUrl[i]* - *targetUrl[i]*).

Implement a function that finds the length of the longest substring *originalUrl* that can be transformed into the corresponding substring in *targetUrl* with a cost that is less than or equal to *cost*.

The function *maxSubstringLength* will take three inputs:
string originalUrl: a string of *n* letters, the original URL
string targetUrl: a string of *n* letters, the target URL
int cost: the maximum allowed cost

The function should return the maximum length of a substring that can be transformed. If no such substring exists, return 0.

Example
originalUrl = "adpgki"
targetUrl = "cdmxki"
cost = 6.

First, let's calculate the cost for each index.

Index	originalUrl Character	targetUrl Character	ASCII Value (Original)	ASCII Value (Target)	Cost
0	a	c	97	99	abs(97 - 99) = 2
1	d	d	100	100	abs(100 - 100) = 0
2	p	m	112	109	abs(112 - 109) = 3
3	g	x	103	120	abs(103 - 120) = 17
4	k	k	107	107	abs(107 - 107) = 0
5	i	i	105	105	abs(105 - 105) = 0

Now evaluate substrings with total cost ≤ 6:

- "adp": Cost = 2 + 0 + 3 = 5 (length = 3)
- "dp": Cost = 0 + 3 = 3 (length = 2)
- "ki": Cost = 0 + 0 = 0 (length = 2)
- Other substrings exceed the cost limit.

The longest valid substring is "adp" with a length of 3. Hence, return 3.

Constraints

- $1 \le n \le 2 * 10^5$
- $0 \le cost \le 10^6$
- The strings *originalUrl* and *targetUrl* contain lowercase English letters only.

▼ Input Format For Custom Testing

The first line contains string *originalUrl*.
The second line contains the string *targetUrl*.
The last line contains an integer, *cost*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
uaccd   →   originalUrl = "uaccd"
gbbeg   →   targetUrl = "gbbeg"
4       →   cost = 4
```

Sample Output

```
3
```

Explanation

First, calculate the cost for each index.

Index	originalUrl Character	targetUrl Character	ASCII Value (Original)	ASCII Value (Target)	Cost
0	u	g	117	103	abs(117 - 103) = 14
1	a	b	97	98	abs(97 - 98) = 1
2	c	b	99	98	abs(99 - 98) = 1
3	c	e	99	101	abs(99 - 101) = 2
4	d	g	100	103	abs(100 - 103) = 3

Now evaluate substrings with total cost ≤ 4:

- "a": Cost = 1 (length = 1)
- "ac": Cost = 1 + 1 = 2 (length = 2)
- "acc": Cost = 1 + 1 + 2 = 4 (length = 3)
- Other substrings exceed the cost limit.

The longest valid substring is "acc" with a length of 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
hffk    →   originalUrl = "hffk"
larb    →   targetUrl = "larb"
3       →   cost = 3
```

Sample Output

```
0
```

Explanation

First, calculate the cost for each index.

Index	originalUrl Character	targetUrl Character	ASCII Value (Original)	ASCII Value (Target)	Cost
0	h	l	104	108	abs(104 - 108) = 4
1	f	a	102	97	abs(102 - 97) = 5
2	f	r	102	114	abs(102 - 114) = 12
3	k	b	107	98	abs(107 - 98) = 9

Since the cost of changing any character exceeds 3, no changes can be made.

Question - 104

Toggle API Endpoint States

A distributed API system manages *n* endpoints to handle incoming requests. The endpoint's states are represented as a binary string, *endpoints,* of length *n*, where an endpoint is active if the character is '1' and inactive if it is '0'. In one operation:

- The state of any endpoint can be toggled, that is, '1' is changed to '0' or vice versa.

An arrangement of *endpoints* is valid if no active endpoint has an inactive endpoint to its immediate right, i.e., the final arrangement does not contain "10" as a substring.

Implement a function that determines the minimum number of operations required to form a valid arrangement of *endpoints*.

The function *getMinimumOperations* takes one input:
string endpoints: A binary string of length *n*, where each character represents the state of an endpoint.

The function should return the minimum number of operations required to make the arrangement valid.

Example
endpoints = "10110".
The initial arrangement "10110" is invalid because:

- The active endpoint at index 0 has an inactive endpoint at index 1 on its immediate right.
- The active endpoint at index 3 has an inactive endpoint at index 4 on its immediate right.

By toggling the states of *endpoints[0]* and *endpoints[4],* the arrangement becomes "00111", which is valid. No two active endpoints have an inactive endpoint to their immediate right. The minimum number of operations required is 2. Hence, return 2.

Constraints

- $1 \le |endpoints| \le 3 * 10^5$

▼ Input Format For Custom Testing

The first line contains a string *endpoints*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
0101    →     endpoints = "0101"
```

Sample Output

```
1
```

Explanation
The initial arrangement "0101" is invalid because:

- The active endpoint at index 1 has an inactive endpoint at index 2 to its immediate right.

Toggle the state of *endpoints[2]* from '0' to '1', and the arrangement becomes "0111", which is valid. No active endpoint has an inactive endpoint to its immediate right. The minimum number of operations required is 1.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
11111    →     endpoints = "11111"
```

Sample Output

```
0
```

Explanation
The initial arrangement is already valid.

Question - 105
Equal Distance

A company tracks the distances traveled by *n* vehicles, represented by the array *distance*. Due to new operational goals, the company wants to standardize the travel distances to match a set of target values.

- In one operation, the distance of a vehicle can either be increased or decreased by 1 unit.

Given *q* target distance requests, represented by the array *targetDistance*.

Implement a function that finds the minimum number of operations required to adjust all vehicle distances to match each target value in *targetDistance[i] (0 ≤ i < q).*

The function *countMinimumOperations* will take two inputs:
int distance[n]: the distances traveled by the vehicles
int targetDistance[q]: the target distances for each adjustment request

The function should return an array of integers that contains the minimum number of operations required to adjust all vehicle distances to match each *targetDistance[i].*

Note: Each target distance adjustment request is independent, meaning the travel distances are restored before processing the next request.

Example:
n = 3
q = 4
distance[] = [50, 55, 60]
targetDistance[] = [60, 55, 50, 65]

1. *targetDistance[0]* = 60 → Required operations = [10, 5, 0] → Total = 15
2. *targetDistance[1]* = 55 → Required operations = [5, 0, 5] → Total = 10
3. *targetDistance[2]* = 50 → Required operations = [0, 5, 10] → Total = 15
4. *targetDistance[3]* = 65 → Required operations = [15, 10, 5] → Total = 30

The answer is [15, 10, 15, 30].

Constraints
- $1 \le n \le 2 * 10^5$
- $1 \le distance[i] \le 10^9$
- $1 \le q \le 2 * 10^5$
- $1 \le targetDistance[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of vehicles.
Each of the next *n* lines contains an integer, *distance[i],* where *0 ≤ i < n.*
The next line contains an integer, *q*, the number of distance adjustment requests.
Each of the next *q* lines contains an integer, *targetDistance[i]*, where *0 ≤ i < q*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        →    distance[] size n = 3
2        →    distance[] = [2, 5, 1]
5
1
3        →    targetDistance[] size q = 3
8        →    targetDistance[] = [8, 4, 3]
4
3
```

Sample Output

```
16
6
5
```

Explanation

1. *targetDistance[0]* = 8. The number of operations required = [6, 3, 7] to adjust all distances to 8. Total operations = 6 + 3 + 7 = 16.
2. *targetDistance[1]* = 4. The number of operations required = [2, 1, 3] to adjust all distances to 4. Total operations = 2 + 1 + 3 = 6.
3. *targetDistance[2]* = 3. The number of operations required = [1, 2, 2] to adjust all distances to 3. Total operations = 1 + 2 + 2 = 5.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
1        →    distance[] size n = 1
2        →    distance[] = [2]
5        →    targetDistance[] size q = 5
8        →    targetDistance[] = [8, 4, 3, 10, 6]
4
3
10
6
```

Sample Output

```
6
2
1
8
4
```

Explanation

1. *targetDistance[0]* = 8. The number of operations required = [6] to adjust all distances to 8. Total operations = 6.
2. *targetDistance[1]* = 4. The number of operations required = [2] to adjust all distances to 4. Total operations = 2.
3. *targetDistance[2]* = 3. The number of operations required = [1] to adjust all distances to 3. Total operations = 1.
4. *targetDistance[3]* = 10. The number of operations required = [8] to adjust all distances to 10. Total operations = 8.
5. *targetDistance[4]* = 6. The number of operations required = [4] to adjust all distances to 6. Total operations = 4.

Question - 106
File Block

A software team is working on a data storage system where a file is represented as binary string *fileBlocks.* A '1' represents a compressed file block, and a '0' represents an uncompressed file block. The task is to rearrange the file blocks so that all compressed blocks (1s) are moved to the end of the string.

- In a single operation, the team can select any compressed block (1) and move it to the right until it reaches the end of the string or another compressed block. The cost of the operation is 1 + (the number of positions moved).

Implement a function that finds the maximum total cost required to rearrange the string so that all compressed blocks (1s) are at the right end.

The function *getMaxCost* takes one input:
string fileBlocks: a binary string where '1' represents a compressed file block and '0' represents an uncompressed block

The function should return the maximum total cost to rearrange the string.

Example
fileBlocks = "110100".

The final string should be "000111". The optimal way to maximize the cost of operations is:

- Swap the second and the third characters at a cost of 1 + 1 = 2. The string becomes "101100"
- Swap the first and the second characters at a cost of 1 + 1 = 2. The string becomes "011100".
- Now, move each one to the end, moving 2 places at a cost of 2 + 1 = 3 each.

The total cost of operations is 2 + 2 + 3 * 3 = 13. Hence the answer is 13.

Constraints

- 1 ≤ |*fileBlocks*|≤ 10^5
- Each character of *fileBlocks* is either '0' or '1'.

▼ Input Format For Custom Testing

A single line that contains a string, *fileBlocks*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
10100    →   fileBlocks = "10100"
```

Sample Output

```
8
```

Explanation
The string to form is "00011".
The optimal way to perform the operations is from "10100" to "01100" to "01001" to "00011" at costs 2, 3, and 3, respectively. Thus the total cost of operations is 2 + 3 + 3 = 8.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
01110    →   fileBlocks = "01110"
```

Sample Output

```
6
```

Explanation
The string to form is "00111".

The optimal order of operations is from "01110" to "01101" to "01011" to "00111" at a cost of 2 each.

Question - 107
Ensemble Model Selection

A machine learning team is building ensembles by selecting models based on their accuracy scores, represented by the array *modelAccuracies*.

A valid ensemble must satisfy the following conditions:

- It contains at least *minModels* models.
- Every model in the ensemble has an accuracy score within the range [*minAccuracy*, *maxAccuracy*] (inclusive).

Implement a function that determines how many valid ensembles can be formed.

The function *countValidEnsembles* takes four inputs:
int modelAccuracies[n]: the accuracy scores of available models
int minModels: the minimum number of models required in an ensemble
int minAccuracy: the minimum allowed accuracy score of a model in an ensemble
int maxAccuracy: the maximum allowed accuracy score of a model in an ensemble

The function should return the number of valid ensembles that can be formed.

Example
modelAccuracies = [12, 4, 6, 13, 5, 10]
minModels = 3
minAccuracy = 4
maxAccuracy = 10

Filter models with accuracy within the range [4, 10]: [4, 6, 5, 10].
Calculate the number of valid ensembles. At least *minModels* = 3 models are required for an ensemble.

- Ensembles of size 3: {4, 5, 6}, {4, 5, 10}, {4, 6, 10}, {5, 6, 10} → 4 ensembles.
- Ensembles of size 4: {4, 5, 6, 10} → 1 ensemble.

The total valid ensembles are 5. Hence, return 5.

Constraints

- $1 \le n \le 20$
- $1 \le minModels \le n$
- $1 \le minAccuracy \le maxAccuracy \le 1000$
- $1 \le modelAccuracies[i] \le 1000$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *modelAccuracies*.
The next *n* lines each contain an element *modelAccuracies[i]*, where *0 ≤ i < n*.
The next line contains an integer, *minModels*, the minimum number of models required in an ensemble.
The next line contains an integer, *minAccuracy*, the lower limit of accuracy scores for models to be included.
The next line contains an integer, *maxAccuracy*, the upper limit of accuracy scores for models to be included.

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-----     --------
4     →   modelAccuracies[] size n = 4
4     →   modelAccuracies = [4, 8, 5, 6]
8
5
6
1     →   minModels = 1
5     →   minAccuracy = 5
7     →   maxAccuracy = 7
```

Sample Output 0

```
3
```

Explanation:

Filter models with accuracy within the range [5, 7]: [5, 6].
Calculate the number of valid ensembles:

- Ensembles of size 1: {5}, {6} → 2 ensembles.
- Ensembles of size 2: {5, 6} → 1 ensemble.

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-----     --------
4     →   modelAccuracies[] size n = 4
4     →   modelAccuracies = [4, 8, 5, 6]
8
5
6
2     →   minModels = 2
5     →   minAccuracy = 5
7     →   maxAccuracy = 7
```

Sample Output 1

```
1
```

Explanation
Filter models with accuracy within the range [5, 7]: [5, 6].
Calculate the number of valid ensembles:

- Ensembles of size 2: {5, 6} → 1 ensemble.

Question - 108
Optimal Scheduling

In a distributed machine learning (ML) system, multiple models need to be trained using *n* graphics processing units (GPUs). Each training job has a specific time window, defined by its *startTimes[i]* and *endTimes[i]* (both inclusive). A single GPU can handle only one job at a time, but the system can utilize multiple GPUs.

Implement a function that determines the minimum number of GPUs required to handle all training jobs, ensuring no two jobs overlap on the same GPU.

The function *getMinGPUs* takes the following inputs:
int startTimes[n]: the start times of training jobs
int endTimes[n]: the end times of training jobs

The function should return the minimum number of GPUs needed to execute all training jobs without overlapping.

Example
n = 3
startTimes = [1, 3, 4]
endTimes = [3, 5, 6]

1. If the system has only one GPU, the first job starts at 1 and ends at 3. The second job starts at 3. Since both jobs need a GPU at 3, they overlap. More than one GPU is required.
2. If the system has two GPUs, the first job runs on the first GPU from 1 to 3, the second job runs on the second GPU from 3 to 5, and the third job runs on the first GPU from 4 to 6.

Thus, a minimum of 2 GPUs is required, and the function should return 2.

Constraints

- $1 \le n \le 10^5$
- $1 \le startTimes[i] \le endTimes[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *startTimes*.
Each of the next *n* lines contains an integer, *startTimes[i]*.
The next line contains an integer, *n*, the number of elements in *endTimes*.
Each of the next *n* lines contains an integer, *endTimes[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
3        →     startTimes[] size n = 3
1        →     startTimes = [1, 2, 3]
2
3
3        →     endTimes[] size n = 3
3        →     endTimes = [3, 3, 5]
3
5
```

Sample Output

```
3
```

Explanation
If using 2 GPUs, the first and second jobs finish at time 3, and the third job starts at time 3. Due to the conflict, at least one more GPU is required.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
3        →     startTimes[] size n = 3
1        →     startTimes = [1, 4, 7]
4
7
3        →     endTimes[] size n = 3
2        →     endTimes = [2, 4, 10]
4
10
```

Sample Output

```
1
```

Explanation
If we use 1 GPU, jobs execute from time 1 to 2, then from time 4 to 4, and finally from time 7 to 10. There are no conflicts.

Question - 109
Search Query Matcher

In an e-commerce product search system, users input queries to search for items. You are given an array *products* of length *n* representing the product names and an array *queries* of length *q* containing search query strings.

Implement a function that, for each query string, returns all products that are anagrams of the query string. An anagram is any string that can be formed by rearranging the letters of another string.

The function *getProductMatches* takes the following inputs:

string products[n]: the list of all available product names
string queries[q]: the list of search query strings

The function should return a list of products for each query that are anagrams of the query string, sorted alphabetically.

Example
n = 4
products = ["duel", "speed", "dule", "cars"]
q = 2
queries = ["spede", "deul"]

- For *queries[0] = "*spede*",* the only anagram of "spede" is "speed". The answer for this query string, sorted alphabetically is ["speed"].
- For *queries[1] = "*deul*",* both the "duel" and "dule" are anagrams of "deul". The answer for this query string, sorted alphabetically is ["duel", "dule"].

Hence, the function should return [["speed"], ["duel", "dule"]].

Constraints
- 1 ≤ *n, q* ≤ 5000
- 1 ≤ length of *products[i]*, length of *queries[i]* ≤ 100
- It is guaranteed that each query word has at least one anagram in the *products* list.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *products*.
Each of the next *n* lines contains a string, *products[i]*.
The next line contains an integer, *q*, the number of elements in *queries*.
Each of the next *q* lines contains a string, *queries[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
6       →      products[] size n = 6
allot   →      products = ["allot", "cat", "peach", "dusty", "act", "cheap"]
cat
peach
dusty
act
cheap
3       →      queries[] size q = 3
tac     →      queries = ["tac", "study", "peahc"]
study
peahc
```

Sample Output

```
act cat
dusty
cheap peach
```

Explanation
- For *queries[0]* = "tac", the anagrams of "tac" sorted in alphabetical order are "act" and "cat".
- For *queries[1]* = "study", the only anagram of "study" is "dusty".
- For *queries[2]* = "peahc", the anagrams of "peahc" sorted in alphabetical order are "cheap" and "peach".

Hence, the function should return [["act", "cat"], ["dusty"], ["cheap", "peach"]].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
5        →     products[] size n = 5
emits    →     products = ["emits", "items", "baker", "times", "break"]
items
baker
times
break
2        →     queries[] size q = 2
mites    →     queries = ["mites", "brake"]
brake
```

Sample Output

```
emits items times
baker break
```

Explanation

- For *queries[0]* = "mites", the anagrams of "mites" sorted in alphabetical order are "emits", "items", and "times".
- For *queries[1]* = "brake", the anagrams of "brake" sorted in alphabetical order are "baker" and "break".

Question - 110
Transaction Segments

In a financial tracking system, *n* transactions are recorded in the array *transactionValues*, where *transactionValues[i]* represents the value of a transaction at time *i*.

Implement a function that counts the number of strictly increasing segments of exactly *k* consecutive transactions.

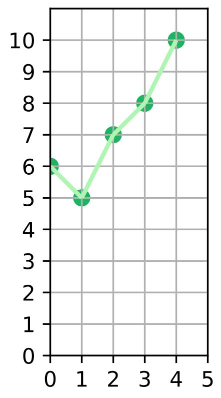
The function *countIncreasingSegments* takes the following inputs:
int transactionValues[n]: the values of each transaction over time
int k: the number of consecutive transactions in each segment

The function should return the number of strictly increasing segments that span exactly *k* consecutive transactions.

Note: A strictly increasing segment refers to a sequence where each transaction value is greater than the previous one.

Example
n = 5
transactionValues = [6, 5, 7, 8, 10]
k = 3

The transaction values are plotted over time, with the X-axis representing the time of the transaction and the Y-axis representing the transaction values at that time.



In this case, there are two strictly increasing segments of exactly *k* = 3 consecutive transactions:

1. Formed by transactions at times 1, 2, and 3 with values 5, 7, and 8, respectively.
2. Formed by transactions at times 2, 3, and 4 with values 7, 8, and 10, respectively.

The answer is 2.

Constraints

- $1 \le k \le n \le 2 \times 10^5$
- $1 \le$ *transactionValues[i]* $\le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in the array *transactionValues*.
Each of the *n* subsequent lines contains an integer that describes the array *transactionValues*.
The last line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

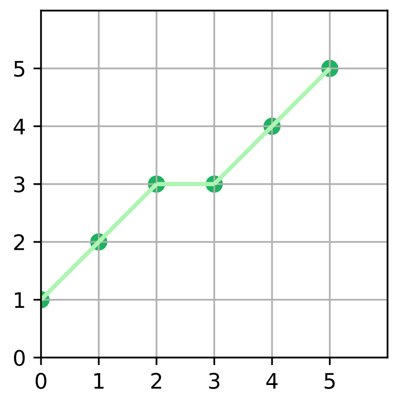
```
STDIN          FUNCTION
-----          --------
6         →    transactionValues[] size, n = 6
1         →    transactionValues = [1, 2, 3, 3, 4, 5]
2
3
3
4
5
3         →    k = 3
```

Sample Output

```
2
```

Explanation

The transaction values are plotted over time, with the X-axis representing the time of the transaction and the Y-axis representing the transaction values at that time.



There are two strictly increasing segments of exactly *k* = 3 consecutive transactions:

1. Formed by transactions at times 0, 1, and 2 with values 1, 2, and 3, respectively.
2. Formed by transactions at times 3, 4, and 5 with values 3, 4, and 5, respectively.

▼ Sample Case 1

Sample Input For Custom Testing

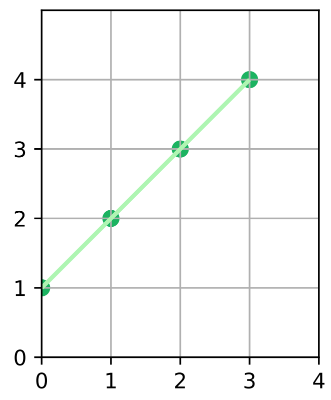
```
STDIN          FUNCTION
-----          --------
4         →    transactionValues[] size, n = 4
1         →    transactionValues = [1, 2, 3, 4]
2
3
4
4         →    k = 4
```

Sample Output

```
1
```

Explanation

The transaction values are plotted over time, with the X-axis representing the time of the transaction and the Y-axis representing the transaction values at that time.



In this case, since *k* is equal to *n*, all transactions form a single, strictly increasing segment.

Question - 111
Merge Optimization

In a data processing system, there are *n* data points, each with an integer value, represented by the array *dataPoints*. The objective is to perform operations on the *dataPoints* until only one data point remains. In one operation:

- Any two elements can be merged to create a new data point whose value is the sum of the two merged points. The computational cost of each operation is the value of the new data point.

Implement a function that computes the minimum possible computational cost to merge all data points into one.

The function *getMinCost* takes the following input:
int dataPoints[n]: the values of the data points

The function should return the minimum possible cost to merge all data points into one.

Example
n = 3
dataPoints = [20, 30, 40]

An optimal sequence is:

1. Select data points with values 20 and 30 and merge them to obtain [50, 40]. The cost is 20 + 30 = 50.
2. Select data points with values 50 and 40 and merge them to obtain [90]. The cost is 50 + 40 = 90.

The total cost for the merges is 50 + 90 = 140, so the function should return 140.

Constraints

- $2 \le n \le 10^5$
- $1 \le dataPoints[i] \le 100$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *dataPoints*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *dataPoints[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
3        ->  dataPoints[] size n = 3
30       ->  dataPoints[] = [30, 10, 20]
10
20
```

Sample Output

```
90
```

Explanation

An optimal series of operations is:

1. Select data points with values 10 and 20 and merge them to obtain [30, 30]. The cost is 10 + 20 = 30.
2. Select data points with values 30 and 30 and merge them to obtain [60]. The cost is 30 + 30 = 60.

The total cost for the merges is 60 + 30 = 90.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
2        ->  dataPoints[] size n = 2
100      ->  dataPoints[] = [100, 1]
1
```

Sample Output

```
101
```

Explanation

Only one merge can be made, combining 100 and 1, with the computational cost of 100 + 1 = 101.

Question - 112
Personalized Grouping

In a recommendation system, an array *engagementScores* of size *n* represents the engagement scores of recommended items.

Implement a function that determines the maximum size of a subset of items that, when their engagement scores are sorted in increasing order, the difference between any two consecutive scores is either 0 or 1.

The function *findMaxGroupSize* takes the following input:

int engagementScores[n]: the engagement scores of each item

The function should return the size of the largest group where the difference between any two consecutive scores is at most 1, when sorted in increasing order.

Example

n = 5

engagementScores = [10, 12, 13, 9, 14]

Any group of size 1 is valid. Several valid groups of size 2 are possible, such as [9, 10] and [12, 13]. The largest group is [12, 13, 14], so return 3.

Constraints

- $1 \le n \le 10^5$
- $1 \le engagementScores[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *engagementScores*.

Each of the *n* subsequent lines contains the elements of the array, *engagementScores*.

▼ Sample Case 0

Sample Input

```
STDIN    Function
-----    --------
4    →    engagementScores[] size n = 4
4    →    engagementScores = [4, 13, 2, 3]
13
2
3
```

Sample Output

```
3
```

Explanation

The largest group is [2, 3, 4].

▼ Sample Case 1

Sample Input

```
STDIN        Function
-----        --------
5     →    engagementScores[] size n = 5
```

```
10    →    engagementScores = [10, 12, 1, 10, 4]
12
1
10
4
```

Sample Output

```
2
```

Explanation

The largest group is [10, 10].

Question - 113

Transaction Sequence Optimization

In a financial portfolio risk management system, *n* transactions are either gains or losses, represented as integers in the array *transactions*. You can rearrange the *transactions* array in any order to maximize the number of transactions during which the portfolio balance stays positive, starting from an initial balance of 0.

Implement a function that determines the maximum number of transactions during which the portfolio balance stays above zero.

The function *getMaxPositiveTransactions* takes the following input:

int transactions[n]: the gains or losses for each transaction

The function should return the maximum number of transactions during which the portfolio balance stays positive.

Example

n = 4

transactions = [1, -20, 3, -2]

An optimal transaction order is [3, -2, 1, -20]. The portfolio balance remains positive for the first three transactions, resulting in a balance of 3 - 2 + 1 = 1. It is not possible to process all 4 transactions while keeping the balance positive.

Hence, the function returns 3.

Constraints

- $1 \le n \le 10^5$
- $-10^9 \le transactions[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *transactions*.

Each of the next *n* subsequent lines contains an integer of the array, *transactions*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
4     →  n = 4
-3    →  transactions = [-3, 0, 2, 1]
0
2
1
```

Sample Output

```
3
```

Explanation
An optimal transaction order is [2, 0, 1, -3]. The portfolio balance remains positive for the first three transactions, resulting in a balance of 2 - 0 + 1 = 3. It is not possible to process all 4 transactions while keeping the balance positive.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     FUNCTION
-----     --------
3      →  n = 3
-3     →  transactions = [-3, 0, -2]
0
-2
```

Sample Output

```
0
```

Explanation
There is no transaction order that results in a positive account balance.

Question - 114
File Access Tracker

In a file management system, *m* users upload various files, each identified by a unique *file_id*.

You are given:

- An array records of size *n*, where each entry *records[i]* is formatted as <user_id> <file_id> <file_path>. This represents a file uploaded by a user, with a unique identifier and its corresponding file path.
- An array queries of size *q*, where each entry contains a *file_id* for which a lookup is requested.

For each query, return:

1. The file path associated with the requested *file_id*.
2. The cumulative request count for the corresponding *user_id* up to the current query.

The function should process the queries in the given order and track how many times each user has been queried.

The function *trackFileAccess* takes the following inputs:
int m: the number of users
string records[n]: the records of file uploads and file paths in no particular order
string queries[q]: the file IDs requested in order

The function should return a 2-D array of strings that, for each query, contains the file path and cumulative request count for the user up to the current query.

Example
n = 3, *m* = 3
records = ["0 sdsf www.google.com", "1 juytf www.google.com", "0 opoit www.kaggle.com"]
q = 3
queries = ["juytf", "sdsf", "opoit"]

Queries (File ID Requested)	Corresponding File Path	User ID	Cumulative request count for the user up to the current query
juytf	www.google.com	1	1
sdsf	www.google.com	0	1

opoit	www.kaggle.com	0	2

Hence, the function should return [["www.google.com", "1"], ["www.google.com", "1"], ["www.kaggle.com", "2"]].

Constraints

- $1 \le n, m, q \le 10^5$
- $0 \le user\_id \le m - 1$
- The length of any file ID does not exceed 5 characters.
- The length of each record is no longer than 30 characters.
- Each file ID/path pair is distinct.
- Each requested file ID is present in the records.

▼ Input Format For Custom Testing

The first line contains an integer, *m*, the number of users.
The next line contains an integer, *n*, the number of strings in *records*.
Each of the next *n* lines contains a string *records[i]*.
The next line contains an integer, *q*, the number of queries.
Each of the next *q* lines contains a string *queries[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN                            FUNCTION
-----                            --------
3                           →    m = 3
4                           →    records[] size n = 4
0 glggl www.google.com      →    records = ["0 glggl www.google.com", "0 fcbok www.facebook.com", "2 lefts
www.hackerrank.com", "0 hckrk www.hackerrank.com"]
0 fcbok www.facebook.com
2 lefts www.hackerrank.com
0 hckrk www.hackerrank.com
2                           →    queries[] size q = 2
hckrk                       →    queries = ["hckrk", "lefts"]
lefts
```

Sample Output

```
www.hackerrank.com 1
www.hackerrank.com 1
```

Explanation

Queries (File ID Requested)	Corresponding File Path	User ID	Cumulative request count for the user up to the current query
hckrk	www.hackerrank.com	0	1
lefts	www.hackerrank.com	2	1

Hence, the function should return [["www.hackerrank.com", "1"], ["www.hackerrank.com", "1"]].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                            FUNCTION
-----                            --------
1                           →    m = 1
4                           →    records[] size n = 4
0 tezox www.google.com      →    records = ["0 tezox www.google.com", "0 rrakl www.facebook.com", "0 crona
www.hackerrank.com", "0 hckrk www.hackerrank.com"]
0 rrakl www.facebook.com
```

```
0 crona www.hackerrank.com
0 hckrk www.hackerrank.com
2                          →    queries[] size q = 2
rrakl                      →    queries = ["rrakl", "crona"]
crona
```

Sample Output

```
www.facebook.com 1
www.hackerrank.com 2
```

Explanation

Queries (File ID Requested)	Corresponding File Path	User ID	Cumulative request count for the user up to the current query
rrakl	www.facebook.com	0	1
crona	www.hackerrank.com	0	2

Hence, the function should return [["www.facebook.com", "1"], ["www.hackerrank.com", "2"]].

Question - 115
Gateway Throttling

In an API rate-limiting system, each incoming request from a domain must be processed while adhering to certain request limits. There is a given array *requests* of size *n*, where *requests[i]* ($0 \le i < n$) represents the domain receiving a request at the i^{th} second.

The gateway allows at most 2 requests from a domain within a 5-second window, and at most 5 requests within a 30-second window.

Implement a function that determines whether each request can be processed. It should return "{status: 200, message: OK}" if the request is allowed, or "{status: 429, message: Too many requests}" if the request exceeds the rate limit.

The function *getDomainRequestStatus* takes the following input:
  string requests[n]: the list of domains making requests to the gateway

The function should return a list of strings representing the status of each request.

Example:
Suppose *n* = 7 and *requests* = ["www.xyz.com", "www.abc.com", "www.xyz.com", "www.pqr.com", "www.abc.com", "www.xyz.com", "www.xyz.com",]

Time	Request	Is Allowed?	Status of Request
0	www.xyz.com	Yes	{status: 200, message: OK}
1	www.abc.com	Yes	{status: 200, message: OK}
2	www.xyz.com	Yes	{status: 200, message: OK}
3	www.pqr.com	Yes	{status: 200, message: OK}
4	www.abc.com	Yes	{status: 200, message: OK}
5	www.xyz.com	Yes	{status: 200, message: OK}
6	www.xyz.com	No -> 2 requests from "www.xyz.com" were accepted within 5 seconds.	{status: 429, message: Too many requests}

Hence the answer is ["{status: 200, message: OK}", "{status: 200, message: OK}", "{status: 200, message: OK}", "{status: 200, message: OK}", "{status: 200, message: OK}", "{status: 200, message: OK}", "{status: 429, message: Too many requests}"].

Constraints

- $1 \le n \le 10^4$
- 1 ≤ length of *requests[i]* ≤ 15
- *requests[i]* consists of lowercase English characters and dot ('.') characters.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *requests*.
Each of the next *n* lines contains a string, *requests[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
 STDIN              FUNCTION
-------             ----------
7                →  requests[] size n = 7
www.abc.com      →  requests = ["www.abc.com", "www.hd.com", "www.abc.com", "www.pqr.com", "www.abc.com",
"www.pqr.com", "www.pqr.com"]
www.hd.com
www.abc.com
www.pqr.com
www.abc.com
www.pqr.com
www.pqr.com
```

Sample Output

```
{status: 200, message: OK}
{status: 200, message: OK}
{status: 200, message: OK}
{status: 200, message: OK}
{status: 429, message: Too many requests}
{status: 200, message: OK}
{status: 429, message: Too many requests}
```

Explanation

- The fifth request is dropped because it exceeds the limit of 2 requests from "www.abc.com" within a 5-second window.
- Similarly, the last request is dropped as it exceeds the limit of 2 requests from "www.pqr.com" within a 5-second window.

▼ Sample Case 1

Sample Input For Custom Testing

```
 STDIN            FUNCTION
-------           ----------
4              →  requests[] size n = 4
www.hr.com     →  requests[] = ["www.hr.com", "www.hr.com", "www.hr.com", "www.hr.com"]
www.hr.com
www.hr.com
www.hr.com
```

Sample Output

```
{status: 200, message: OK}
{status: 200, message: OK}
{status: 429, message: Too many requests}
{status: 429, message: Too many requests}
```

Explanation

The first two requests are allowed by the gateway. Requests from "www.hr.com" are blocked for the next 3 seconds due to exceeding the rate limit within the allowed time window.

Question - 116
Endpoint Monitoring

In an API management system, there are *n* endpoints, with IDs "s1", "s2", ..., "sn", where each endpoint corresponds to a unique server. The system maintains a list *logEntries*, of size *m*, each in the format "<endpoint_id> <status>", where status is either "success" or "error".

If an endpoint logs three consecutive "error" statuses, it is deactivated and replaced with a new endpoint of the same ID.

Implement a function that determines how many times endpoints were replaced with a new endpoint.

The function *countReplacedEndpoints* takes the following inputs:
  int n: the number of endpoints
  string logEntries[m]: a list of strings representing the status of each endpoint

The function should return the number of times endpoints were replaced.

Example
Suppose *n = 2*, *m = 6, logEntries* = ["s1 error", "s1 error", "s2 error", "s1 error", "s1 error", "s2 success"]

logEntries	Endpoint ID	Status	Last 3 statuses of this particular Endpoint	Is the Endpoint Replaced?
s1 error	s1	error	[error]	No
s1 error	s1	error	[error, error]	No
s2 error	s2	error	[error]	No
s1 error	s1	error	[error, error, error]	Yes
s1 error	s1	error	[error]	No
s2 success	s2	success	[]	No

The endpoint having ID = "s1" was replaced one time with a new endpoint of the same ID. No other endpoints were replaced.
Hence, the answer is 1.

Constraints
- $1 \le n \le 200$
- $1 \le m \le 2 \times 10^4$

▼ Input Format For Custom Testing

The first line contains an integer, *n*.
The next line contains an integer, *m*.
Each of the next *m* lines contains a string, *logEntries[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN              FUNCTION
-----              --------
5          →       n = 5
7          →       m = 7
s1 error   →       logEntries = ["s1 error", "s2 error", "s1 error", "s4 success", "s5 error", "s3 success", "s1
error"]
s2 error
```

```
s1 error
s4 success
s5 error
s3 success
s1 error
```

Sample Output

```
1
```

Explanation
Only the endpoint with ID = "s1" was replaced. It had three errors without any successes at the first, third, and last lines of the log.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
3         →     n = 3
6         →     m = 6
s2 error  →     logEntries = ["s2 error", "s3 error", "s2 error", "s2 error", "s3 error", "s3 error"]
s3 error
s2 error
s2 error
s3 error
s3 error
```

Sample Output

```
2
```

Explanation
The endpoints with IDs "s2" and "s3" were replaced at the fourth and last log entries respectively.

Question - 117
Optimizing Inventory Turnover

A warehouse has *n* product batches, each with a turnover potential given by the array *turnover*. The goal is to reach at least *target* units of total turnover in the fewest number of days.

Each day, the warehouse can process any subset of batches. However, due to logistical delays, the contribution of each batch decreases based on its processing order that day:

- The first batch processed contributes its full value.
- The second contributes one unit less.
- The third contributes two units less, and so on.
- A batch contributes *max(0, turnover value - its position in the day's order (0-based))*.

Implement a function to find the minimum number of days required to meet or exceed the target sales of *target* units.

The function *getMinDaysToMeetSales* will take two inputs:
int turnover[n]: the turnover of each batch
int target: the sales target

The function should return the minimum number of days required to meet or exceed the target sales of *target* units. If achieving the target sales after processing all batches is impossible, the function should return -1.

Example
n = 5
turnover = [2, 3, 1, 1, 2]

target = 8

An optimal way of achieving the target sales is (consider 0-based indexing for batches)
On day 1, we process the batch at index 1, contributing 3.
On day 2, we process the batch at index 4, bringing the total to 5.
On day 3, we process the batch at index 0, increasing the total to 7.
Finally, on day 4, we process the batches at indices 2 and 3, achieving the target of 8.

The minimum number of days required to meet or exceed the target sales is 4.

Constraints

- $1 \leq n \leq 2*10^3$
- $1 \leq turnover[i] \leq 2*10^3$
- $1 \leq target \leq 4*10^6$

▼ Input Format For Custom Testing

The first line contains an integer *n*, the size of the array *turnover*.
Each of the next *n* lines contains an integer *turnover[i]*.
The next line contains an integer, *target*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
5      →   turnover[] n = 5
5      →   turnover = [5, 5, 5, 5, 5]
5
5
5
5
10     →   target = 10
```

Sample Output

```
1
```

Explanation
On Day 1, process the first three batches, with contributions 5, 4, and 3, respectively. The total sales reach 5+4+3=12, exceeding the target (10). Hence, the optimal answer is 1.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
2      →   turnover[] n = 2
2      →   turnover = [2, 3]
3
10     →   m = 10
```

Sample Output

```
-1
```

Explanation
With two batches of turnover [2, 3] and a target of 10, the maximum achievable sales are 5, which is less than the target. Hence, it is impossible to meet the target, output -1.

Question - 118
Minimum Bandwidth Boosts

Given *n* servers in a data center, each initially having a bandwidth of 0, and an array, *target*, where *target[i]* represents the desired bandwidth for the i^{th} server.

- To increase bandwidth, you can perform a boost operation, which increments the bandwidth of all servers in a range *[l, r]* by 1 where $1 \le l \le r \le n$.

Implement a function to find the minimum boost operations required so that the bandwidth of each server is exactly equal to its corresponding target value.

The function *getMinimumBoosts* will take one input:
int target[n]: An array where *target[i]* represents the required bandwidth that the i^{th} server must attain.

The function should return an integer denoting the minimum boost operations required so that the bandwidth of each server is exactly equal to its corresponding target value.

Example
target = [1, 2, 2, 1]

Bandwidths start at [0, 0, 0, 0]. The following operations can be performed to achieve the target bandwidths:

1. Boost range [0, 2], updating the bandwidths to [1, 1, 1, 0].
2. Boost range [2, 3], updating the bandwidths to [1, 2, 2, 1].

Thus, the minimum number of boost operations required is 2.

Constraints
- $1 \le n \le 10^5$
- $0 \le target[i] \le 10^9$

▾ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *target*.
Each of the next *n* lines contains an integer *target[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN          FUNCTION
-----          --------
3        →     target[] size n = 3
1        →     target = [1, 3, 2]
3
2
```

Sample Output 0

```
3
```

Explanation
The following operations can be performed to achieve the target bandwidths:

1. Boost range [0, 2], updating the bandwidths to [1, 1, 1].
2. Boost range [1, 2], updating the bandwidths to [1, 2, 2].
3. Boost range [1, 1], updating the bandwidths to [1, 3, 2].

Thus, the minimum number of boost operations required is 3.

▾ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
2        →    target[] size n = 2
1        →    target = [1, 2]
2
```

Sample Output 1

```
2
```

Explanation

The following operations can be performed to achieve the target bandwidths:

1. Boost range [0, 1], updating the bandwidths to [1, 1].
2. Boost range [1, 1], updating the bandwidths to [1, 2].

Thus, the minimum number of boost operations required is 2.

Question - 119
Optimizing Servers

You manage a data center with *n* servers, each with two attributes: computing power and operational cost. The i^{th} server has a computing power of *power[i]* and an operational cost of *cost[i]*. To ensure the optimal use of infrastructure, inefficient servers are to be discarded.

Server *i* is better than server *j* if both of the following conditions are true:

- *power[i]* > *power[j]* (server *i* is more powerful than server *j*)
- *cost[i]* < *cost[j]* (server *i* is cheaper to operate than server *j*)

Implement a function that determines the number of servers that remain after removing all inefficient servers.

The function *optimizeServers* takes the following input:
  int power[n]: the computing powers of the servers
  int cost[n]: the operational costs of the servers

The function should return an integer, the number of servers that remain after removing all inefficient servers.

Note: The elements in *power[]* and *cost[]* are distinct.

Example
power = [1, 2, 3]
cost = [1, 4, 2]

The example represents three servers:
Server 1: computing power 1, operational cost 1
Server 2: computing power 2, operational cost 4
Server 3: computing power 3, operational cost 2
Server 2 can be discarded as its power is less than that of server 3 and costs more than server 3.
As for servers 1 and 3, neither is better, thus, we are left with two servers.

Constraints

- $1 \le n \le 2*10^5$
- $0 \le power[i] < 10^9$
- $0 \le cost[i] < 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *power*.
Each of the next *n* lines contains an integer *power[i]*.

The next line contains an integer *n*, the size of the array *cost*.

Each of the next *n* lines contains an integer *cost[i]*.

▾ Sample Case 0

Sample Input 0

```
STDIN         FUNCTION
-----         --------
4       ->    power[] size n = 4
1       ->    power = [1,4,6,3]
4
6
3
4       ->    cost[] size n = 4
1       ->    cost = [1,2,4,3]
2
4
3
```

Sample Output 0

```
3
```

Explanation

We start with 4 servers [(1, 1),(4, 2),(6, 4),(3, 3)], representing (power, cost) for each server.

Server 4, (3, 3), is dominated by server 2, (4, 2) (higher power and lower cost).

The remaining servers are (1, 1),(4, 2),(6, 4), so the answer is 3.

▾ Sample Case 1

Sample Input 1

```
STDIN         FUNCTION
-----         --------
5       ->    power[] size n = 5
1       ->    power = [1,2,3,4,5]
2
3
4
5
5       ->    cost[] size n = 5
1       ->    cost = [1,2,3,4,5]
2
3
4
5
```

Sample Output 1

```
5
```

Explanation

We start with 5 servers [(1, 1), (2, 2), (3, 3), (4, 4), (5, 5)], representing (power, cost) for each server.

No server dominates any other since no server has both higher power and lower cost.

Hence, the answer is 5.

Question - 120
Bandwidth Contention Analysis

A streaming platform allocates bandwidth to handle *n* user requests. Each request starts at time *start[i]* and ends at *end[i]* (inclusive). The bandwidth contention is defined as the highest number of user requests the platform handles at the same time.

Implement a function that determines the earliest time when the platform experiences the maximum bandwidth contention.

The function *getMaxBandwidthTime* will take two inputs:
int start[n]: the start times of user requests
int end[n]: the end times of user requests

The function should return the earliest time when the platform experiences the maximum bandwidth contention.

Example
Suppose *start* = [1, 6, 2 ,9] and *end* = [8, 7, 6, 10]

Time	Connection	Active Requests
1	Request 1 Begins	1
2	Request 3 Begins	1, 3
3		1, 3
4		1, 3
5		1, 3
6	Request 2 Begins	1, 2, 3
7	Request 3 Ends	1, 2
8	Request 2 Ends	1
9	Request 1 Ends and Request 4 Begins	4
10		4
11	Request 4 Ends	

The maximum bandwidth contention is 3 which happens first at the 6th second.

Constraints

- $1 \le n \le 10^5$
- $1 \le start[i] \le end[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the number of bandwidth requests.
Each of the next *n* lines contains an integer *start[i]*.
The next line contains the same integer *n,* the number of bandwidth requests.
Each of the next *n* lines contains an integer *end[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN       FUNCTION
-----       --------
5       →   start[] size n = 5
```

```
2       →   start[] = [2, 3, 7, 4, 7]
3
7
4
7
5       →   end[] size n = 5
4       →   end[] = [4, 5, 8, 7, 10]
5
8
7
10
```

Sample Output 0

```
4
```

Explanation

At time *t = 4,* the platform processes the first, second, and fourth requests. At time *t = 7,* the platform processes the last three requests.

The earliest time when the bandwidth contention reaches 3 requests is t=4.

▼ Sample Case 1

Sample Input 1

```
STDIN       FUNCTION
-----       --------
3       →   start[] size n = 3
1       →   start[] = [1, 1, 1]
1
1
3       →   end[] size n = 3
5       →   end[] = [5, 5, 5]
5
5
```

Sample Output 1

```
1
```

Explanation

All requests are being processed by the platform at *t = 1.*

Question - 121

Username Strength Checker

In a secure messaging system, there are *n* usernames stored in a string array *usernames[n]* and a dictionary *commonWords[m]* containing *m* common words.

The usernames' strengths need to be assessed. A username is classified as "weak" if any of the following conditions is true:

- The username is in the dictionary.
- Any substring of the username is a word in the dictionary.
- It contains only digits 0 through 9.
- It consists of all uppercase or all lowercase letters.
- It is shorter than 6 characters.

Otherwise, the username is "strong".

Implement a function that evaluates each username's strength according to the criteria.

The function *getUsernameStrength* will take two inputs:
  string usernames[n]: the usernames to check
  string commonWords[m]: commonly used words

The function should return a list of strings, with "weak" or "strong" as the classification for each username.

Example

Suppose *n* = 5, *m* = 3, *usernames* = [“iliketoCoDe”, “teaMAKEsmehappy”, “abracaDabra”, “pasSword”, “blackcoffeeISthebest”], *commonWords* = [“coffee”, “coding”, “happy”]

username	strong/weak	Remarks
iliketoCoDe	strong	
teaMAKEsmehappy	weak	contains "happy"
abracaDabra	strong	
pasSword	strong	
blackcoffeeISthebest	weak	contains "coffee"

Constraints

- $1 \le n \le 10^3$
- $1 \le m \le 10^5$
- 1 ≤ |*usernames[i]*| ≤ 20
- 1 ≤ |*commonWords[i]*| ≤ 20
- The usernames consist of lowercase, uppercase, and numeric characters only.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *usernames*.
Each line *i* of the *n* subsequent lines contains a string, *usernames[i].*
The next line contains an integer, m, the number of elements in *commonWords*.
Each line *i* of the m subsequent lines contains a string, *commonWords*[i].

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        →    n = 3
hello    →    usernames[] = ["hello", "chargeR", "pass123"]
chargeR
pass123
5        →    m = 5
hello    →    commonWords[] = ["hello", "123", "password", "xyz", "999"]
123
password
xyz
999
```

Sample Output

```
weak
strong
weak
```

Explanation

username	strong/weak	Remarks

hello	weak	contains "hello"
chargeR	strong	
pass123	weak	contains "123"

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN            FUNCTION
-----            --------
3          →     n = 3
12345      →     usernames[] = ["12345", "YUIOYES", "qwertyuiop"]
YUIOYES
qwertyuiop
2          →     m = 2
ty         →     commonWords[] = ["ty", "xyz"]
xyz
```

Sample Output

```
weak
weak
weak
```

Explanation

username	strong/weak	Remarks
12345	weak	only contains numbers, too short
YUIOYES	weak	only contains uppercase letters
qwertyuiop	weak	only contains lowercase letters

Question - 122
Classify Texts

A classification system evaluates if given *n* texts are spam based on a list of *k* spam words.

A text is labeled as "spam" if it contains at least two spam words. Each occurrence of a spam word in a text counts towards the total. The spam words are not case-sensitive.

Implement a function that determines whether each text is "spam" or "not_spam".

The function *classifyTexts* will take two inputs:
string *texts[n]*: the texts to evaluate
string *spamWords[k]*: the spam words

The function should return an array of *n* strings, either "spam" or "not_spam" for each text.

Example
Suppose texts = [“free prize worth millions”, “ten tips for a carefree lifestyle”] and *spamWords* = [“free”, “money”, “win”, “millions”].

Text	Spam Words	Answer
free prize worth millions	free, millions	spam

ten tips for a carefree lifestyle	-	not_spam

Hence the answer is ["spam", "not_spam"].

Constraints

- $1 \le n \le 10^3$
- $1 \le k \le 10^5$
- $1 \le |texts[i]| \le 10^5$
- $1 \le |spamWords[i]| \le 10^5$
- It is guaranteed that the texts and spam words consist of lowercase and uppercase English letters and spaces only.
- The combined length of all *spamWords* does not exceed 10^7.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *texts*.
Each of the next *n* lines contains a string, *texts[i]*.
The next line contains an integer, *k*, the number of elements in *spamWords*.
Each of the next *k* lines contains a string, *spamWords[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN                     FUNCTION
-----                     --------
2                   →     texts[] size n = 2
I paid him paid     →     texts = ["I paid him paid", "Summertime Sadness"]
Summertime Sadness
3                   →     spamWords[] size k = 3
I                   →     spamWords = ["I", "Sadness", "paid"]
Sadness
paid
```

Sample Output

```
spam
not_spam
```

Explanation

The first text contains three spam words: 2 "paid" and an "I." The second text contains only a single spam word: "Sadness."

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                        FUNCTION
-----                        --------
2                      →     texts[] size n = 2
Let it go              →     texts= ["Let it go", "The right thing to do"]
The right thing to do
5                      →     spamWords[] size k = 5
to                     →     spamWords = ["to", "do", "right", "go", "let"]
do
right
go
let
```

Sample Output

```
spam
spam
```

Explanation

The first text contains two spam words: "go", and "let". Spam words are not case-sensitive. The second one contains three spam words: "to", "do", and "right".

Question - 123
Training Timer

In a multi-model machine learning system, different models are trained sequentially on a single GPU. If a higher-priority task arrives during training, the lower-priority task is paused until the higher-priority task finishes.

There are *n* different AI models to be trained on the GPU, each with a unique ID between *0* and *n-1*. A list of *m* training logs is represented as an array of strings, *logs[m]*. Log entries follow the format: {modelId}:{start/end}:{timestamp}, indicating that the model with ID = modelId*,* either starts or ends at a time identified by the *timestamp* value. If any model is introduced while the previous one is running, the previous model is put on hold, and the current model is trained until it is completed or put on hold.

Implement a function that determines each model's training time.

The function *getTotalTrainingTime* will take two inputs:
int *n*: the number of models train
string *logs[m]*: each string is a training log

The function should return an array specifying the exclusive training times of each model.

Example
Suppose *n* = 3, *logs* = ["0:start:0", "2:start:4", "2:end:5", "1:start:7", "1:end:10", "0:end:11"]

Timestamp	Model Running	Remarks
0	0	Model 0 starts
1	0	
2	0	
3	0	
4	2	Model 0 is paused and Model 2 starts
5	2	Model 2 ends
6	0	Model 0 resumes
7	1	Model 0 is paused and Model 1 starts
8	1	
9	1	
10	1	Model 1 ends
11	0	Model 0 ends

Thus, the total number of seconds allocated to Models 0, 1, and 2 are 6, 4, and 2, respectively. Hence the answer is [6, 4, 2].

Constraints
- 1 ≤ *n* ≤ 100
- 1 ≤ *m* ≤ 500

- 0 ≤ *model_id* < n
- 0 ≤ *timestamp* ≤ 3 * 10^3
- The timestamps are given in non-decreasing order.
- No two starting timestamps and no two ending timestamps are equal.
- Every model's "start" call has a corresponding "end" call.

▼ Input Format For Custom Testing

The first line contains an integer, *n*.
The next line contains an integer, *m*, the size of *logs[]*.
Each of the next *m* lines contains a string, *logs[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
2           →   n = 2
4           →   logs[] size m = 4
0:start:0   →   logs = ["0:start:0","1:start:3","1:end:6","0:end:10"]
1:start:3
1:end:6
0:end:10
```

Sample Output

```
7
4
```

Explanation
The training of the model happens in the following order:
Time = [0, 2]: model ID = 0 trains on the GPU.
Time = [3, 6]: model ID = 1 trains on the GPU.
Time = [7, 10]: model ID = 0 trains on the GPU.
Thus the exclusive times for different models are as follows:
model (ID = 0) = 3 + 4 = 7
model (ID = 1) = 4

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
3           →   n = 3
6           →   logs[] size m = 6
0:start:0   →   logs = ["0:start:0","1:start:3","1:end:6","2:start:8","2:end:10","0:end:12"]
1:start:3
1:end:6
2:start:8
2:end:10
0:end:12
```

Sample Output

```
6
4
3
```

Explanation
The training of models happens in the following order:
Time = [0, 2]: model ID = 0 trains on the GPU.
Time = [3, 6]: model ID = 1 trains on the GPU.
Time = [7, 7]: model ID = 0 trains on the GPU.
Time = [8, 10]: model ID = 2 trains on the GPU.

Time = [11, 12]: model ID = 0 trains on the GPU.
Thus the exclusive times for different models are as follows:
model (ID = 0) = 3 + 1 + 2 = 6
model (ID = 1) = 4
model (ID = 2) = 3

Question - 124
Optimizing Resource Utilization

In a ML model pipeline, *n* tasks must be executed concurrently on available resources. Each task requires exactly one resource and has a specific time window defined by its start time *start[i]* and end time *end[i]* (inclusive).

Implement a function that determines the minimum number of resources required to handle tasks concurrently, ensuring no two tasks overlap on the same resource.

The function *getMinResources* will take two inputs:
int start[n]: the start times of the tasks
int end[n]: the end times of the tasks

The function should return the minimum number of resources needed to handle all tasks concurrently without any overlap.

Example
Suppose *n* = 5, *start* = [1, 8, 3, 9, 6], *end* = [7, 9, 6, 14, 7].

An optimal schedule requires 3 resources. The times in parentheses represent the inclusive start and end times for each task:

- Resource 1: [(1, 7), (8, 9)]
- Resource 2: [(3, 6), (9, 14)]
- Resource 3: [(6, 7)]

Return 3.

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le start[i] \le end[i] \le 10^9$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the number of tasks.
Each of the next *n* lines contains an integer *start[i]*.
The next line contains the same integer *n,* the number of tasks.
Each of the next *n* lines contains an integer *end[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN       FUNCTION
-----       --------
5       →   start[] size n = 5
2       →   start[] = [2, 1, 5, 5, 8]
1
5
5
8
5       →   end[] size n = 5
5       →   end[] = [5, 3, 8, 6, 12]
3
```

```
8
6
12
```

Sample Output 0

```
3
```

Explanation

An optimal schedule:

- Resource 1: [(1,3), (5,8)]
- Resource 2: [(2,5), (8, 12)]
- Resource 3: [(5,6)]

▼ Sample Case 1

Sample Input 1

```
STDIN       FUNCTION
-----       --------
4       →   start[] size n = 4
2       →   start[] = [2, 2, 2, 2]
2
2
2
4       →   end[] size n = 4
5       →   end[] = [5, 5, 5, 5]
5
5
5
```

Sample Output 1

```
4
```

Explanation

The optimal schedule:

- Resource 1: [(2, 5)]
- Resource 2: [(2, 5)]
- Resource 3: [(2, 5)]
- Resource 4: [(2, 5)]

Question - 125

Token Validation

A UI class validation tool identifies valid tokens from a given string *tokenString*. Tokens are separated by a space, and *valid* tokens must meet the following criteria:

- Consists of at least 3 characters.
- Contains only alphanumeric characters (letters A-Z in upper or lower case and numbers 0-9).
- Includes at least one vowel (a, e, i, o, u) and at least one consonant (all other letters).

Implement a function that counts the number of valid tokens present in *tokenString*.

The function *countValidTokens* will take one input:
  string tokenString: a string to identify valid tokens in

The function should return the number of valid tokens present in *tokenString*.

Example

Suppose *tokenString* = "This is an example string 234".

Token	Is Valid	Reason
This	Yes	At least 3 characters, contains a vowel and a consonant
is	No	Fewer than 3 characters
an	No	Fewer than 3 characters
example	Yes	At least 3 characters, contains a vowel and a consonant
string	Yes	At least 3 characters, contains a vowel and a consonant
234	No	Does not contain a vowel or a consonant

Constraints

- $1 \le |tokenString| \le 10^5$.
- Every token in string *tokenString* is separated by a space.
- *tokenString* can contain all ASCII characters.

▼ Input Format For Custom Testing

The first line contains a string, tokenString.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN                                   FUNCTION
-----                                   --------
This is Form16 submis$ion date       →  tokenString = "This is Form16 submis$ion date"
```

Sample Output

```
3
```

Explanation

Only 'This', 'Form16', and 'date' are valid tokens. Since 'is' only contains 2 characters and 'submis$ion' has a character that is not alphanumeric, they are not valid.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                                FUNCTION
-----                                --------
Alex wins the game       →            tokenString = "Alex wins the game"
```

Sample Output

```
4
```

Explanation

All the tokens are valid.

Question - 126
Algorithm Comparison

In a product recommendation system, two key-value pairs, *predictions1,* and *predictions2*, are given as strings. They represent the product recommendations made by two different algorithms.

Implement a function to find the keys where the recommended products (values) differ between the two algorithms. If a key is present in only one algorithm's recommendations, it should not be considered.

The function *getPredictionDiff* takes two inputs:
  string *predictions1*: product recommendations made by Algorithm 1
  string *predictions2*: product recommendations made by Algorithm 2

The function should return a list of the keys where the recommendations differ between the two algorithms, sorted alphabetically.

Example
predictions1 = '{"hello":"world","hi":"hello","you":"me"}'
predictions2 = '{"hello":"world","hi":"helloo","you":"me"}'

The only common key where the values differ is "hi". Hence the answer is ["hi"].

Constraints
- 1 ≤ |*predictions1*|, |*predictions2*| ≤ 10^5. Every key is distinct in each of the predictions.
- There is no whitespace in the strings.

▼ Input Format For Custom Testing

The first line contains a string, *predictions1*.
The next line contains a string, *predictions2*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN                                              FUNCTION
-----                                              --------
{"hacker":"rank","input":"output"}        →     predictions1 = {"hacker":"rank","input":"output"}
{"hacker":"ranked","input":"wrong"}       →     predictions2 =
{"hacker":"ranked","input":"wrong"}
```

Sample Output

```
hacker
input
```

Explanation
Recommendations are different for both keys so they are returned in alphabetical order.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                                              FUNCTION
-----                                              --------
{"hacker":"rank","input":"output"}      →      predictions1 = {"hacker":"rank","input":"output"}
{"hacker":"rank","input":"output"}      →      predictions2 = {"hacker":"rank","input":"output"}
```

Sample Output

```

```

Explanation
Recommendations match for both keys, so return an empty list.

Question - 127
Profit Tracker

A system for a retail platform is being developed to track the profit and loss of products based on transactions and price updates. You are given an array, *events* of size *n*, where each event is categorized into one of the following:

1. BUY <product> <quantity>: Indicates that the retail platform has purchased <quantity> units of <product> at the current price.

2. SELL <product> <quantity>: Indicates that the retail platform has sold <quantity> units of <product> at the current price.
3. CHANGE <product> <price>: Indicates that the price of <product> has changed by <price> amount, which can be positive (price increase) or negative (price decrease).
4. QUERY: Requests the net profit/loss from the start of all transactions and price changes up to the current point in time.

Implement a function that calculates the net portfolio profit/loss values for all "QUERY" events in the order they appear.

The function *getNetProfit* will take one input:

string *events[n]*: the events to process

The function should return an array of net portfolio profit/loss values for all "QUERY" events in the order they appear.

Example

n = 7

events = ["BUY headphones 20", "BUY laptop 50", "CHANGE headphones 6", "QUERY", "SELL laptop 10", "CHANGE laptop -2", "QUERY"]

Events	Portfolio	Current Portfolio Profit/Loss
BUY headphones 20	headphones 20	0
BUY laptop 50	headphones 20 laptop 50	0
CHANGE headphones 6	headphones 20 laptop 50	120
QUERY		120
SELL laptop 10	headphones 20 laptop 40	120
CHANGE laptop -2	headphones 20 laptop 40	40
QUERY	headphones 20 laptop 40	40

The answer is [120, 40].

Constraints

- $1 \le n \le 10^5$, where *n* is the size of the array *events*.
- $1 \le |events[i]| \le 21$
- For a SELL event, it is guaranteed that the inventory contains enough units of the product.
- $1 \le$ quantity $< 10^3$
- The absolute value of a change in the price of any product at any event will not exceed 10^3.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *events*.

Each of the *n* subsequent lines contains element of the array *events*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
2           →   events[] size n = 2
BUY hackr 2 →   events = [“BUY hackr 2", “QUERY”]
QUERY
```

Sample Output

```
0
```

Explanation
The retail platform purchased 2 products of hackr, then had a "QUERY" event. Since there is no change in product price, there is 0 profit so far. Hence the answer is [0].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                 FUNCTION
-----                 --------
6                 →   events[] size n = 6
BUY stock2 2      →   events = ["BUY stock2 2", "BUY stock1 4", "CHANGE stock2 -8", "SELL stock1 2", "BUY stock3
3", "QUERY"]
BUY stock1 4
CHANGE stock2 -8
SELL stock1 2
BUY stock3 3
QUERY
```

Sample Output

```
-16
```

Explanation
The price of 2 units of stock2 decreased by 8. Hence the answer is [-16]

Question - 128
Prediction Cache Retrieval

In a machine learning model, there are *n* data entries stored in the model's output cache *cacheEntries.* Each entry contains a *timestamp*, a *modelId*, and a *predictionValue*. The *timestamp* indicates when the prediction was made, *modelId* identifies the machine learning model, and *predictionValue* is the model's output as an integer represented as a string. Note that the *modelId* can repeat across entries.

The query handler receives *q* query strings in the array *queries[q][2]*, where each query is of the form {*modelId*, *timestamp*}. The *modelId* is the machine learning model's identifier, and *timestamp* represents the time the prediction was made.

Implement a function that determines the *predictionValue* for each query based on the given *modelId* and *timestamp*.

The function *getQueryAnswers* takes two inputs:
string cacheEntries[n][3]: the cache data entries.
string queries[q][2]: the queries.

The function should return an array of integers that represent the *predictionValue* for each query based on the given *modelId* and timestamp.

Example
n = 3
cacheEntries = [["12:30:22", "a2er5i80", "125"], ["09:07:47", "io09ju56", "341"], ["01:23:09", "a2er5i80", "764"]]
q = 2
queries = [["a2er5i80", "01:23:09"], ["io09ju56", "09:07:47"]]

- The *queries[0]* refers to the entry where the model a2er5i80 made a prediction at 01:23:09, with the *predictionValue* of 764.
- The *queries[1]* refers to the entry where the model io09ju56 made a prediction at 09:07:47, with the *predictionValue* of 341.

Event Type	Data	Cache	Query Result
Update cache	["09:07:47", "io09ju56", "341"]	io09ju56 = 341	-

Query	["io09ju56", "09:07:47"]	io09ju56 = 341	341
Update cache	["12:30:22", "a2er5i80", "125"]	io09ju56 = 341, a2er5i80 = 125	-
Update Cache	["01:23:09", "a2er5i80", "764"]	io09ju56 = 341, a2er5i80 = 764	-
Query	["a2er5i80", "01:23:09"]	io09ju56 = 341, a2er5i80 = 764	764

Return [764, 341].

Constraints

- $1 \le n \le 10^5$
- $1 \le q \le 10^5$
- $1 \le int(cacheEntries[i][2]) \le 10^8$
- *cacheEntries[i][0]* represents a valid timestamp in the format hh:mm: ss
- *size(cacheEntries[i][0])* = 8
- *cacheEntries[i][1],* is an alphanumeric value consisting of only lowercase English letters (a-z), and digits (0-9)
- It is guaranteed that the queried {*modelId*, *timestamp*} pair is present in the cache
- There are no duplicate timestamps for a particular *modelId.*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *cacheEntries.*

The next line contains a constant integer, 3.

Each of the next *n* lines contains three space-separated strings, *cacheEntries[i][0], cacheEntries[i][1], cacheEntries[i][2].*

The next line contains a constant integer, *q*, the number of elements in *queries.*

The next line contains a constant integer, 2.

Each of the next *q* lines contains two space-separated strings, *query[i][0]* and *query[i][1].*

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN                        FUNCTION
-----                        --------
3                       →    cacheEntries [] size n = 3
3                       →    constant integer = 3 (elements per cacheEntries[i])
03:00:23 1e3fg08w 101   →    cacheEntries = [["03:00:23", "1e3fg08w", "101"],
12:45:36 ef41t561 243                       ["12:45:36", "ef41t561", "243"],
21:09:31 ko0e51j8 809                       ["21:09:31", "ko0e51j8", "809"]]
1                       →    queries[] size q = 1
2                       →    constant integer = 2 (elements per queries[i])
ef41t561 12:45:36       →    queries = ["ef41t561", "12:45:36"]
```

Sample Output

```
243
```

Explanation

The query corresponds to the prediction entry at index 1, with a value = 243.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                        FUNCTION
-----                        --------
4                       →    cacheEntries[] size n = 4
3                       →    constant integer = 3 (elements per cacheEntries[i])
01:34:05 166k1kph 352   →    cacheEntries = [["01:34:05", "166k1kph", "352"],
56:38:37 8pvj20oo 107                       ["56:38:37", "8pvj20oo", "107"],
36:17:33 r0v06eec 180                       ["36:17:33", "r0v06eec", "180"],
20:34:20 e15y6dv4 490                       ["20:34:20", "e15y6dv4", "490"]]
2                       →    queries[] size q = 2
```

```
2                          →    constant integer = 2 (elements per queries[i])
e15y6dv4 20:34:20          →    queries = [["e15y6dv4", "20:34:20"],
8pvj20oo 56:38:37                          ["8pvj20oo", "56:38:37"]]
```

Sample Output

```
490
107
```

Explanation

The queries correspond to prediction entries at indices 3 and 1, respectively, with values 490 and 107.

Question - 129

Maximize Server Performance

In a cloud infrastructure system, there are *n* servers, each with a performance score and activation status, represented by the arrays *performanceScores* and *activationStatus.* If the activation status is 1, the server is active, and 0 indicates an inactive server.

Implement a function that calculates the maximum possible total performance value of active servers by activating at most *k* consecutive servers.

The function *getMaxPerformanceSum* takes the following inputs:

int performanceScores[n]: the performance scores of the servers

int activationStatus[n]: the activation status of the servers

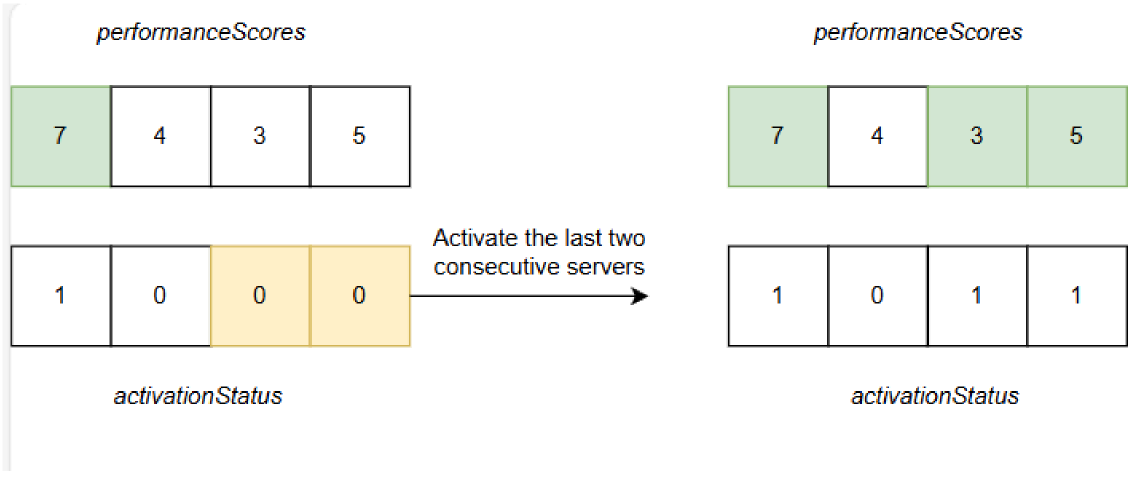
int k: the maximum number of consecutive servers that can be activated

The function should return the maximum possible total performance score sum by activating the servers optimally.

Note: A subarray is any contiguous segment of the array.

Example

Given *n* = 4, *k* = 2, *performanceScores* = [7, 4, 3, 5], and *activationStatus* = [1, 0, 0, 0]



The answer is the sum of the activated serves after performing the activation optimally: 7 + 3 + 5 = 15.

Constraints

- 1 ≤ *k* ≤ *n* ≤ 10^5
- 1 ≤ *performanceScores [i]* ≤ 10^4
- 0 ≤ *activationStatus[i]* ≤ 1

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in the *performanceScores*.
Each of the next *n* lines contains an integer, *performanceScores[i]*.
The next line contains an integer, *n*, the number of elements in *activationStatus*.
Each of the next n lines contains an integer, *activationStatus[i]*.
The next line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
6        →     performanceScores[] size n = 6
1        →     performanceScores = [1 , 3 , 5 , 2 , 5 , 4]
3
5
2
5
4
6        →     activationStatus[] size n = 6
1        →     activationStatus = [1 , 1 , 0 , 1 , 0 , 0]
1
0
1
0
0
3        →     k = 3
```

Sample Output

```
16
```

Explanation
Activate the servers at indices 2 through 4, i.e., [1, 1, **0, 1, 0**, 0], to get [1, 1, **1, 1, 1**, 0]. The performance score sum is 1 + 3 + 5 + 2 + 5 = 16.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
5        →     performanceScores[] size n = 5
1        →     performanceScores = [1 , 3 , 2 , 3 , 6]
3
2
3
6
5        →     activationStatus[] size n = 5
0        →     activationStatus = [0 , 0 , 0 , 1 , 0]
0
0
1
0
2        →     k = 2
```

Sample Output

```
9
```

Explanation
Activate the servers at indices 3 and 4, i.e., [0, 0, 0, **1, 0**], to get [0, 0, 0, **1, 1**]. The performance score sum is 3 + 6 = 9.

Question - 130
Data Cleaning

In a data cleaning pipeline, engineers optimize text datasets by alternately removing substrings: Alex removes substrings with an odd number of vowels in the first step, followed by Chris removing substrings with an even number of vowels in the next step.

Alex always goes first, then the engineers take turns optimally until no valid substring remains. The goal is to determine which engineer removes the last substring.

Implement a function *optimizeDataCleaning* to determine the outcome of the cleaning process for each dataset, assuming alternating and optimal cleaning.

The function *optimizeDataCleaning* takes the following input:
   string datasets[n]: each string represents a dataset

The function should return an array of strings specifying the result of the cleaning process for each dataset.

Note: Vowels are 'a', 'e', 'i', 'o', and 'u'.

Example
Given *n = 2* and *datasets* = ["git", "dry"]

1. For the *datasets[0]* = "git", Alex removes the entire string, leaving it empty. Since no valid moves are left, Chris cannot make a move, and Alex performs the final removal.
2. For the *datasets[1]* = "dry", the string contains no vowels. Since Alex cannot make any moves, Chris is considered to have made the final removal.

Hence the answer is ["Alex", "Chris"].

Constraints
- 1 ≤ *n* ≤ 100
- 1 ≤ *|datasets[i]|* ≤ 10^5
- All strings in the *datasets* consist of lowercase English letters only.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *datasets*.
Each of the next *n* lines contains a string, *datasets[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
3        →     datasets[] size n = 3
lgzpc    →     datasets = ["lgzpc", "lchxlo", "xrwzg"]
lchxlo
xrwzg
```

Sample Output

```
Chris
Alex
Chris
```

Explanation
Chris takes the first and third datasets which contain no vowels. Since Alex cannot make a move, Chris is considered to have made the final valid removal.
In the second dataset, *datasets[1] = "lchxlo"*, there is exactly one vowel 'o'. Alex removes the entire string.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
```

```
1       →     datasets[] size n = 1
cim     →     datasets = ["cim"]
```

Sample Output

```
Alex
```

Explanation

Alex removes the entire string since it contains an odd number of vowels.